LIBRARY NAVAL POSTGRADUATE SCHOOL
MONTEREY, CA 93940

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

AN APPROACH FOR IMPLEMENTING A MICROCOMPUTER BASED REPORT ORIGINATION SYSTEM IN THE ADA PROGRAMMING LANGUAGE

by

Michael Richard Critz

March 1983

Thesis Co-Advisor: H. Titus
Thesis Co-Advisor: G. R. Porter

Approved for Public Release; Distribution Unlimited

T207853

LIBRARY, NAVAL POSTGRADUATE SCHOOL
MONTEREY, CA 93940

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)
An Approach for Implementing a Microcomputer Based Report Origination System in the Ada Programming Language

5. TYPE OF REPORT & PERIOD COVERED
Master's Thesis;
March 1983

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)
Michael Richard Critz

8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Naval Postgraduate School
Monterey, California 93940

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS
Naval Postgraduate School
Monterey, California 93940

12. REPORT DATE
March 1983

13. NUMBER OF PAGES
296

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)
Naval Postgraduate School
Monterey, California 93940

15. SECURITY CLASS. (of this report)
Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)
Approved for Public Release; Distribution Unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

report origination system
formatted reports
Ada
microcomputer
operational reports

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This thesis examines the use of an inexpensive commercial microcomputer for the preparation of Naval Reporting Structure Operational Reports. These highly formatted reports provide critical unit information used by the National Command Authority and the Joint Chiefs of Staff in assessing the nation's defense posture. Since these reports are processed by a computer, correct formatting and data entry are essential to preserve the timeliness and accuracy of the information. The requirements of a Report Origination

20. (cont'd)

System are investigated from the perspective of the system operator, the message drafter and the message releasing authority. Interfaces are developed which provide for system application to different hardware configurations. A subset of the Ada language is used to allow structured programming and data abstraction techniques. Elements of the Unit Status and Identity Report (UNITREP) are implemented using this method.

Approved for Public Release; Distribution Unlimited

An Approach for Implementing a Microcomputer Based Report Origination System in the Ada Programming Language

by

Michael Richard Critz
Lieutenant, United States Navy
B.S., Rensselaer Polytechnic Institute, 1975

Submitted in partial fullfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
March 1983

ABSTRACT

This thesis examines the use of an inexpensive commercial microcomputer for the preparation of Naval Reporting Structure Operational Reports. These highly formatted reports provide critical unit information used by the National Command Authority and Joint Chiefs of Staff in assessing the nation's defense posture. Since these reports are processed by computer, correct formatting and data entry are essential to preserve the timeliness and accuracy of the information. The requirements of a Report Origination System are investigated from the perspective of the system operator, the message drafter and the message releasing authority. Interfaces are developed which provide for system application to different hardware configurations. A subset of the Ada language is used to allow structured programming and data abstraction techniques. Elements of the Unit Status and Identity Report (UNITREP) are implemented using this method.

TABLE OF CONTENTS

I. BACKGROUND -- 11

A. THE PROBLEM -- 11

B. EARLY EFFORTS -- 13

1. Composite Report (COMPREP) --------------- 13

2. X/C 13 Increment I (COMPREP) ------------- 14

3. Composite Operations Reporting System (CORS) 14

4. Report Origination System (ROS) ---------- 15

C. RECENT DEVELOPMENTS --------------------------- 16

1. Navy Reporting Structure (NRS) ----------- 17

2. Error Study ----------------------------------- 18

3. Casualty Report (CASREP) Test ------------ 18

4. Word Processing Approach ----------------- 21

II. DESIGN OBJECTIVES ------------------------------- 23

A. USER CONSIDERATIONS --------------------------- 23

B. DESIGN REQUIREMENTS --------------------------- 26

C. HARDWARE CONSIDERATIONS ---------------------- 28

D. SOFTWARE CONSIDERATIONS ---------------------- 30

III. SYSTEM OVERVIEW --------------------------------- 31

A. LOGICAL INFORMATION GROUPING ----------------- 31

B. PROGRAM ACTIONS -------------------------------- 33

C. PROGRAM DISPLAYS ------------------------------ 37

D. DATA STORAGE ---------------------------------- 40

IV. IMPLEMENTATION -- 41
  A. ADA SUBSET RESTRICTIONS ------------------------- 41
  B. PROGRAM OVERVIEW ---------------------------------- 44
  C. BASIC PROGRAM SUPPORT COMPONENTS ------------ 50
    1. Console Package (consio) ----------------- 50
    2. Printer Package (printio) ---------------- 56
    3. Global Package (urglobal) ---------------- 57
    4. Utility Package (urutil) ----------------- 69
    5. Test Package (urtest) -------------------- 71
  D. PROGRAM DATA COMPONENTS ---------------------- 72
    1. Free Text Data (urgltl) ------------------ 72
    2. Local Information Data (urlocalA) -------- 73
    3. Administrative Information Data (uradminB) 73
    4. Air Information Data (urairB) ------------ 74
    5. Composite Data (unitrepA/B) -------------- 77
  E. PROGRAM CONTROL COMPONENTS ------------------- 77
    1. Primary Control (unitrep) ---------------- 77
    2. Module Control (unitrep(x)) ------------ 80
    3. File Handling (filerA/B) ----------------- 81
  F. PROGRAM OPERATIVE COMPONENTS ----------------- 89
    1. Common Elements -------------------------- 90
    2. General Modification Constructs ---------- 91
    3. Local Information (urlocal, urlocal1) ---- 95
    4. Administrative Information (uradmin) ----- 96
    5. Air Information (urair, urair1/2/3) ------ 96

G. MISCELLANEOUS ---------------------------------- 98

V. CONCLUSIONS -- 100

APPENDIX A : ROS UNITREP OPERATOR MANUAL ------------ 105

APPENDIX B : JANUS/ADA LIBRARY MODULES ------------- 154

APPENDIX C : RCS UNITREP COMPUTER LISTING ----------- 159

LIST OF REFERENCES ----------------------------------- 294

BIBLIOGRAPHY --- 295

INITIAL DISTRIBUTION LIST ---------------------------- 296

LIST OF TABLES

1-1. Formatted Message Error Study Analysis ----------- 19

1-2. Average Time Delay for Report Messages in Error -- 19

3-1. UNITREP Data Set/Information - Situation Correspondence ------------------------------------- 34

4-1. Terminal Function - ASCII Character Code Correspondence ------------------------------------- 52

4-2. Format Parameter Specifications ------------------ 58

4-3. Error Messages ------------------------------------ 66

4-4. Air Data Storage Requirements -------------------- 75

A-1. Data Item - Valid Access Code Correspondence ----- 124

A-2. Data Set - Screen Display Correspondence --------- 128

LIST OF FIGURES

3-1. ROS UNITREP SUBSYSTEM Overview ------------------- 32

3-2. Report Generation Cycle ------------------------- 38

4-1. Basic Support Organization ---------------------- 46

4-2. Data Structure Organization --------------------- 47

4-3. Program Structure for 'A' Data ------------------ 48

4-4. Program Structure for 'B' Data ------------------ 49

4-5. Print Action Decision Tree ---------------------- 62

4-6. PACKAGE unitrep Action/Area Selection Flow Chart 82

4-7. PACKAGE unitrep Control Flow Chart -------------- 83

4-8. PACKAGE unitrep1 Control Flow Chart ------------- 84

4-9. PACKAGE unitrep2 Control Flow Chart ------------- 86

4-10. PACKAGE unitrep3 Control Flow Chart ------------ 88

A-1. ROS UNITREP SUBSYSTEM Sign On Display ----------- 113

A-2. ROS UNITREP Action Menu Display ----------------- 114

A-3. ROS UNITREP User Prompts ------------------------ 116

A-4. ROS UNITREP Situation Menu Display -------------- 118

A-5. ROS UNITREP Free Text Set Display --------------- 126

A-6. ROS UNITREP Local Infrequent Display ------------ 129

A-7. ROS UNITREP Local Message Display --------------- 132

A-8. ROS UNITREP Message Addressee Display ----------- 133

A-9. ROS UNITREP Local Miscellaneous Display --------- 134

A-10. ROS UNITREP Administrative Display ------------- 136

A-11. ROS UNITREP Air Authorizations Display --------- 138

A-12. ROS UNITREP Air Location Display ---------------- 140

A-13. ROS UNITREP Air Status Display ------------------ 141

I. BACKGROUND

The need for current and correct command and control information within the military has long been recognized. There will most likely never be a time when the services possess assets that are considered in excess of those required to perform the assigned missions. With the advance of technology over the past forty years we have seen a continuing decrease in the time required to act or react to a given situation. The advent of the computer provided both the cause and the means to reduce decision making time. In these days when "rapid deployment" has become a keyword, the ability of the higher authority to quickly and precisely judge the disposition and status of forces has become paramount.

A. THE PROBLEM

To obtain such an appraisal of forces, hundreds of bits of information (normally submitted via message reports) relating to location, personnel manning and training, equipment status and overall readiness condition, for many individual units must be examined.

At the transmitting end we have the originator who must sift through volumes of instructions to determine WHAT is required to be reported, WHEN it is required to be reported

and HOW it is to be reported. All too often the drafting process reverts to examining past reports for the general form and content. Thus errors tend to perpetuate themselves.

At the receiving end, the World Wide Military Command and Control System (WWMCCS) supports the National Command Authority and the Joint Chiefs of Staff by processing, correlating and presenting the information. The computers on which the WWMCCS is based, for all their speed and precision, can only respond to that information for which they are programmed. Although that programming may be arbitrarily complex (to include such items as look-up tables, alternative spellings/formats and sophisticated deduction algorithms) it is still finite and cannot possibly cover all situations. When the computer cannot determine the logical content of a received report it must queue that report, awaiting a human operator´s intervention and subsequent correction. The operator, when manually processing the report, MAY be able to deduce what was meant to be reported, or may be forced to guess. It is not reasonable to assume that the operator will always correctly determine what was intended by an originating unit.

Such is the problem : The computer receiving the information requires correct format and content in order to provide accurate and timely information; the human sending the information does not easily communicate in computer terms, tending more towards a free format, uncoded style.

B. EARLY EFFORTS

It should be clear that the computer is both the protagonist and the antagonist in this situation. It CAN provide current and correct information to the decision maker IF it receives the correct information in the correct format in the first place. However, if the information is not proper in format or content, correcting it will at best create a time late problem and at worst result in contamination of the WWMCCS database with incorrect interpetations. Concluding that computerized information processing was the best means of satisfying the stringent time requirements considering the voluminous data, several approaches were attempted. All depended on the originator sending accurate, timely, properly formatted information.

1. Composite Report (COMPREP)

A Composite Reporting System (COMPREP) was designed and tested by Commander First Fleet in 1971. This was an attempt to reduce the number of required operational reports by substituting a unified report formatted for automatic data processing. With the limited resources available the system was not sufficiently developed to prove its worth as an effective management tool. The main deficiency proved to be the coded format which made the report difficult for even the originator to decipher once the report had been prepared.

2. X/C 13 Increment I (COMPREP)

This study, concluded in 1975, set as its goal the combination of the Naval Status of Forces Report (NAVFORSTAT), Movement Report (MOVEREP), Casualty Report (CASREP) and Emergency Military Standard Requestioning and Issue Procedures (MILSTRIP) into a single reporting system. A separate reporting structure and training for its use were developed. This system was not recommended because it did not totally integrate required operational reports, the messages were not easily drafted and the final message was not easily understood by the originator. The test and evaluation (T & E) report concluded that it was basically a rehash of the the existing system with little genuine improvement. [Ref. 1]

3. Composite Operations Reporting System (CORS)

Reflecting on the conclusions of the COMPREP T & E report, the goals of COMPREP were re-evaluated. This effort, designated the Composite Operations Reporting System, was conducted between the Office of the Chief of Naval Operations (OPNAV) CORS Steering Committee and the Naval Electronics Systems Command (NAVELEX). The goals of the CORS were:

-- to provide alternatives for a reporting system which would provide timely and accurate data to required levels of command

-- to minimize reporting requirements of individual units by combining Employment Schedule, Movement Reports, Casualty Reports and Naval Status of Forces Reports

-- to provide significant improvements in the readability, draftability and communications systems impact.

Of twelve alternatives proposed, the final recommendation was to develop a system in which the originator would manually draft simple, readable, formatted messages using predefined forms and decision logic trees. This alternative was judged most feasible due to its ability to be implemented early. The benefit of a computer based Report Origination System (ROS) in the drafting of the messages was realized but not recommended due to economic constraints. Other alternatives rejected were strict narrative reports with manual insertion into the database by operators at the receiving sites and limited narrative reports with a special front end text processor at the receiving site. These alternatives could not reasonably meet schedule, technology or economic contraints at that time. [Ref. 2]

4. Report Origination System (ROS)

The concept of a computer based system to support the originator in the preparation of formatted reports had been explored during the CCRS effort. Such a system was conceived to have the following properties/capabilities:

-- be microcomputer based

-- be menu driven

-- provide a screen display similar in format to the required report

-- perform error checking in such areas as proper alphanumeric characters, field length and logical content,

-- provide a hard copy output of the required report in acceptable message format.

In December 1976 Holyoak [Ref. 3] of the Naval Postgraduate School issued a thesis regarding the implementation of the ROS for a NAVFORSTAT on a microcomputer system. This ROS was coded in Intel Corporation's PL/M systems language for an 8080 based Intellec - 8 mainframe with 16K of main memory. Subsequently, in June 1977, Godley [Ref. 4] completed a follow-on thesis based on the same hardware but expanded to include general formatted reports.

C. RECENT DEVELOPMENTS

Following the COPS report, several different areas were investigated to gather the information necessary to revamp the operational reporting system. These efforts concentrated on defining the desired data, conducting a study to determine the source of errors in the current system, conducting a ROS prototype development and test and an examination of systems available to the fleet.

1. Navy Reporting Structure (NRS)

As previously stated, one of the goals of the CORS was to develop an operations reporting system which was unified, easy to understand in both concept and output and which provided a simple message drafting capability. Consequently, the Chief of Naval Operations, OP-643, was designated as the central point of contact and approval authority for message text formatting and data element issue and activities within the Department of the Navy. Intense effort resulted in the February 1982 promulgation of the proposed OPNAVINST 3503.x series: Navy Reporting Structure Operational Reports [Ref. 5]. This seven volume instruction consists of the following:

OPNAVINST 3503.1 NRS - General Instructions
OPNAVINST 3503.2 NRS - Staff Report (STAFFREP)
OPNAVINST 3503.3 NRS - Employment Schedule Report (EMSREP)
OPNAVINST 3503.4 NRS - Casualty Report (CASREP)
OPNAVINST 3503.5 NRS - Unit Status and Identity Report (UNITREP)
OPNAVINST 3503.6 NRS - Surface Movement Report (SURFMOVE)
OPNAVINST 3503.7 NRS - Submarine Movement Report (SUBMOVE)

OPNAVINST 3503.1 summarizes the NRS operational reports, sets responsibilities of cognizant commands, details the management of the program and establishes clear guidelines relating to formatting procedures and message preparation. All other instructions in the series relate to specific reports and provide reporting requirements, data set and field descriptions, look up tables and sample worksheets for

preparation of certain report items. Numerous examples are provided throughout the instructions. The influence of human readability is seen in the care with which data field descriptors were selected. The influence of computer processing is seen in the high degree of formatting required.

2. Error Study

Concurrently with the NRS development, NAVELEX conducted a study to determine the source of errors within current formatted reports sent to an error queue and the result of such errors on the timeliness of information entered into the WWMCCS database [Ref. 6]. CASREPs, MOVEREPs UNITREPs and RAINFORMs were examined and catalogued by number/type errors, type unit and time late to the database. 72 percent of all MOVREPs, 61 percent of all UNITREPs and 46 percent of all RAINFORMS were found to be in error. No CASREP error analysis was made. Table 1-1 lists a breakdown of the average errors These averages represent 76.1 percent of the total errors discovered. The mean time delay from transmission to entry into the data base for a message sent to the error queue is shown in table 1-2.

3. Casualty Report (CASREP) Test

Based on the results of the error study the decision was made to develop a prototype BCS system for the NRS CASREP (then pending release as NWP 7 Revision A

Table 1-1. Formatted Message Error Study Analysis

DATA CONTENT ERRORS	PERCENT OF ERRORS
Character Type	2.1
Calculations/Associations	14.3
Message Identification	4.7
Value Mismatch	22.6
Date-Time Group	3.8
Position	1.2
FORMAT/STRUCTURE ERRRORS	
Descriptor	9.9
Sequence	5.7
Missing Mandatory	5.4
Logical Relationships	1.8
Size	3.9
Duplicates	0.7

Table 1-2. Average Time Delay for Report Messages in Error

TYPE MESSAGE	TOTAL TIME DELAY
MCVREP	3 hrs 54 min
UNITREP	3 hrs 45 min
RAINFORM	7 hrs 44 min
CASREP	33 hrs 22 min

Appendix B). The contractor, Sterling Data Applications Incorporated of Sterling VA., programmed the ROS in the BASIC language on an SDS 420 microcomputer system. In addition to providing a hard copy message output, the system also prepared a punched paper tape compatible with communications equipment aboard ship. This prevented typing errors during message center preparations for transmittal. The system was deployed aboard the USS CALIFORNIA (CGN-36) from 1 April 1981 through 3 June 1981 for a ROS Prototype Development Test. Unfortunately NWP 7 Rev A, App. B was not released in time for the test which necessitated review under the then current and proposed instructions. Dual message logs were maintained for the 81 messages processed under the existing and proposed systems. Error checking was accomplished after deployment by comparing these logs and conducting an error queue analysis for CASREPs from the CALIFORNIA.

The test noted that commercial hardware could function successfully in the shipboard environment. Only one hardware failure was noted - a read/write disk head disengagement during sea state 4 (10-15 degrees roll, 5-7 degrees pitch, 4-8 foot waves) operations. This was corrected by positioning the microcomputer at an angle to the ships roll axis. Four presumed software failures occured on the afloat system. However none of these failures could be recreated at the test site ashore. The report implied that the afloat

failures were due to operator error. One software failure occured at the shore site and was corrected. [Ref. 7]

4. Word Processing Approach

The decision to implement OPNAVINST 3503.x, Naval Reporting Structure Operation Reports, has been delayed pending replacement or recoding of the Honeywell 6000 computers on which the WWMCCS resides (projected 1987). In the meantime the Navy had decided on the Xerox 860 Word Processor as the standard word processor approved for service purchase. Current efforts are aimed at providing the fleet a fill-in-the-blank format capability for CASREP and EMSKED reports as soon as possible. As a fill-in-the-blank capability does not completely meet the full requirements of a ROS system (no data cross checking, error notification, etc.), Sterling Data Applications is examining the feasibility of installing a CP/M board in the Xerox 860 to convert it to a true microcomputer. Initial indications are that this is a possible, though costly alternative. Additionally, SDA has recoded the prototype ROS CASREP from the BASIC language to the COBOL language in compliance with DOD standards.

The purpose of the ROS is clear - overcome the boundry of the man-computer interface by providing the the men at sea the tools to easily, rapidly and accurately; draft, prepare, review and transmit formatted reports to higher authority ashore.

Considerable effort has gone into proving that the concept is viable and effective. With the program restricted by current budgetary constraints and actual implementation dependent on the installation of the new WWMCCS computers or recoding of the current system to accept the new reports, it was decided to examine the ROS in a future context. The recent increase in the capabilities of commercial microcomputer equipment and software tools, and the concurrent decrease in cost, appears to make the planned implementation of the ROS concept all the more desirable.

II. DESIGN OBJECTIVES

The purpose of the ROS is to reduce errors in operational reports submitted from the unit level. The concept behind the ROS is to provide the originator a microcomputer based system which will simplify the report generation cycle. The proposed benefits of the ROS are: a greater confidence in the accuracy of the data retrieved from the WWMCCS database by higher command; quicker insertion of update information into the database; reduction of manpower required to correct erroneous reports at the WWMCCS sites; a lowered administrative burden on the originating unit. Three primary areas should be addressed when considering the design of such a system: the user, the hardware on which the system is to be based and the software tools with which the system is to be implemented.

A. USER CONSIDERATIONS

In any report generation cycle several ′user′ distinctions may be made (e.g. clerk/typist, drafter, releaser). Each ′user′ has particular requirements from any system providing assistance in report generation.

1. System Operator

At the most fundamental level, the system operator (i.e. the clerk/typist) is responsible for preparation of

the final report for transmission and the maintenance of historical files. His responsibilities would entail data entry, data storage and retrieval, and an understanding of basic system capabilities. System operator requirements and questions would entail:

- -- How easy is the system to learn and operate?
- -- Are frequent references to a user's manual required?
- -- How 'crash' resistant/reliable is the system?
- -- What is the is the initialization procedure?
- -- How is data entry accomplished?
- -- What is the source of the data to be entered?
- -- What is the input error correction procedure?
- -- Does the system provide a properly formatted output?
- -- Does the system maintain past information? If so, is that information easy to access?
- -- Does the system make the job easier?

2. Drafter

The report drafter must determine when conditions require a report to be submitted. Pertinent information must be gathered, usually from several different sources - one of which may be previously reported data. The proper format for that information is normally decided by the governing instruction. The drafter may be only minimally concerned with the operation of the system or may desire the capability to generate a report in real-time (i.e. without the assistance of the system operator and without a drafting

document). Drafter requirements and questions would be slightly different:

-- How closely does the system correlate to the instruction that requires the report?

-- Does the system provide detailed drafting information (data fields, field lengths, sources of information and requirements for submission)?

-- Does the system provide a means of presenting all information as currently reported?

-- Can the system 'back-up' to any stage in the reporting cycle and present information that was current as of that time (e.g. when required to verify information as requested by higher authority)?

-- How easy is it to learn the system operator's job?

-- Is there sufficient information available, on line at the terminal, to assist in the real time preparation of a report?

-- Does the system make the job simpler/easier?

3. Releaser

The releaser is not concerned with the operation of the system, but primarily directs his attention to the requirements to submit the report, the content of what is being reported and the ability to understand the report with minimal reference to the governing instruction. Releaser requirements:

-- Certainty that the report conforms to the required format.

-- A simple guide which is a decoded version of the report, providing a logical grouping of information and what that information represents.

B. DESIGN REQUIREMENTS

With the three separate 'users' to consider and the variety of their requirements it is clear that a three tiered approach is necessary to provide the individual user the required features.

1. System Operator

The system operator's requirements tend to center about the microcomputer system and the RCS program. To satisy these requirements:

-- Provide a detailed User's Manual covering such areas as installation procedures, general operation, displays, etc.

-- Logically group data sets together in screen displays which are annotated in plain language with adequate on screen prompts. The screen displays should closely follow any drafting document.

-- The system should be user friendly/fault tolerant. There should be no abnormal terminations. Any termination requires user notification and procedures to be followed to resume normal operation.

-- Incorrect entries should trigger clear, concise error messages which inform the user of the error and, when appropriate, the proper action to be taken.

-- Limit keyboard input to only those characters acceptable for Optical Character Reader (OCR) and BAUDCT message transmission codes. All illegal keyboard inputs should be rejected with notification to the user.

-- Data entry should be menu driven in cases where only distinct choices are possible.

-- Where the user must enter data into fields the input should be limited to the allowed field length.

-- If the entry is obtained from a table, the entry should be cross checked against that table to ensure its validity.

- -- Fields of fixed formats such as date-time groups, latititude-longitude positions should appear as a template over which the operator enters the proper data. All such fields (and associated checksums) should be verified.

- -- Numeric fields should be verified and range checked.

- -- Sum/total numeric fields should be cross checked against the sum of the component fields.

- -- Mutually exclusive items should be treated as such within the system and displays.

- -- Input of unit-type specific items should only be allowed for the unit-types authorized (e.g. if only a ship can report some item, an air unit would not be allowed to report it).

- -- The system should provide a two format output option: DD-173 (OCR) and Standard Naval Message Format as received during fleet broadcasts.

2. Drafter

The drafter is obviously concerned with assistance in the preparation of the report. To provide this service the system should:

- -- Prepare a hardcopy rough drafting form. Items on the form should be keyed to the display screens presented to the system operator and therefore be logically grouped. The drafting form should closely follow the instruction requiring the report. Information required on the form will be:
  - -- the data set name, associated field names and a fill in area of the appropriate field width
  - -- source of the information
  - -- meaning associated with each field if not self explanatory
  - -- conditions for submission of the data set
  - -- all currently reported information (in message format)

-- Prepare the proper format internally. The drafter should only provide the necessary information to the system operator for entry into the ROS.

Additionally, there is the constraint of the drafter operating the system for real-time report generation. For this requirement the drafter will require all the assistance of the system operator plus:

-- An additional on line help facility which contains information normally available only in the draft document.

3. Releaser

With automatic formatting, the releaser need only be concerned with the content of the report. Expanding the rough drafting document to include explanations of coded items and allowing an option for preparation of this form based on the information to be reported should satisfy this requirement.

C. HARDWARE CONSIDERATIONS

As the ROS should only be considered a tool in the preparation of reports, it should be implemented on as economical a microcomputer system as possible. It should not require a system that is MIL-SPEC compatible (such as the UYK-20). Off the shelf commercial equipment appears the best approach. Unfortunately the Navy has not standardized purchase procedures/requirements for a general purpose commercial microcomputer as it has for word processors. The

proliferation of commercial models thoughout the fleet is well known - TRS-80s, APPLEs and HPs to name several.

The most common eight-bit microprocessor architecture in use today is that of the Z-80/8080 group. To have the greatest commonality with this group it was determined that the microcomputer should be Z-80 based. Additionally it was desired to have dual single-sided single-density disk drives to provide a minimum of one-half megabyte of storage. This would allow the program executable code to be contained on one disk while data storage files for past reports could be filed on the other disk. Terminal/keyboard groups are as widely varied as microcomputer systems if not more so. The requirement was for a full ASCII keyboard with a minimum 23 line by 79 column terminal display. Other differences between terminals can normally be accomodated by a software interface as long as the electrical connection is via the industry standard RS-232. It was desired that the printer be of the page feed variety in order to develop the necessary procedures for the alignment and individual paging mechanisms as would be used in preparing an OCR Joint Messageform (DD-173).

On the basis of availibility, an ACS 8000-1 Altos Computer System and a DATAMEDIA ELITE 2500 keyboard and terminal were selected. A page printer, however, was not readily available and a Teletype Corporation Model 40 continuous line printer was substituted.

D. SOFTWARE CONSIDERATIONS

Since the most common operating system for commercial microcomputers is currently Digital Research's Control Program for Microprocessors (CP/M), this system was included whenever an operating system was required. Additionally many of the language compilers examined for the system required CP/M.

As it was desired to code the system in a structured language which allowed data abstraction (looking forward to a full system development in the proposed Department of Defense standard language ADA [Ref. 8]), several variants of PASCAL were examined including UCSD Pascal, JRT Pascal, SORCIM Pascal and MP+ Pascal (Digital Research). Initial programming began with UCSD Pascal but was abandoned when it was discovered that the available version did not possess the facilities for separate module compilation. Coding resumed with JRT Pascal but was quickly abandoned for lack of helpful debugging diagnostics. The examination of SORCIM or MP+ as an alternative was interrupted by the ready availability of JANUS/ADA by RR Software, Madison, WI. [Ref. 9]. JANUS/ADA is a limited subset of the proposed standard but met the requirements for a structured language that allowed data abstraction, required CP/M and ran on a Z-80 based system. The limitations imposed by the subset will be refered to as they applied during the implementation of the system (Chapter IV).

III. SYSTEM OVERVIEW

To support the requirements discussed in Chapter II several different computer systems levels must be addressed. At the most elementary level, software interfaces to terminals and printers are required. As the size of the program expanded and it became necessary to physically separate the program into modules a means of module communication was required. The majority of the system is concerned with the capabilities which the program provides and how the report information is grouped, displayed and maintained. The NRS UNITREP [Ref. 10] was selected as the demonstration vehicle. Figure 3-1 details the system overview.

A. LOGICAL INFORMATION GROUPING

Based on the requirements of the three users discussed in Chapter II, one of the driving design factors for the RUS UNITREP is a logical grouping of information - both in the crafting document and screen displays. Upon examination of the UNITREP instruction it was observed that eight areas, called situations, are deliniated which provide a partial logical grouping [Ref. 10: para. 1.5a]:

(1) Personnel Status
(2) Administrative Status
(3) Unit Combat Readiness Assessment

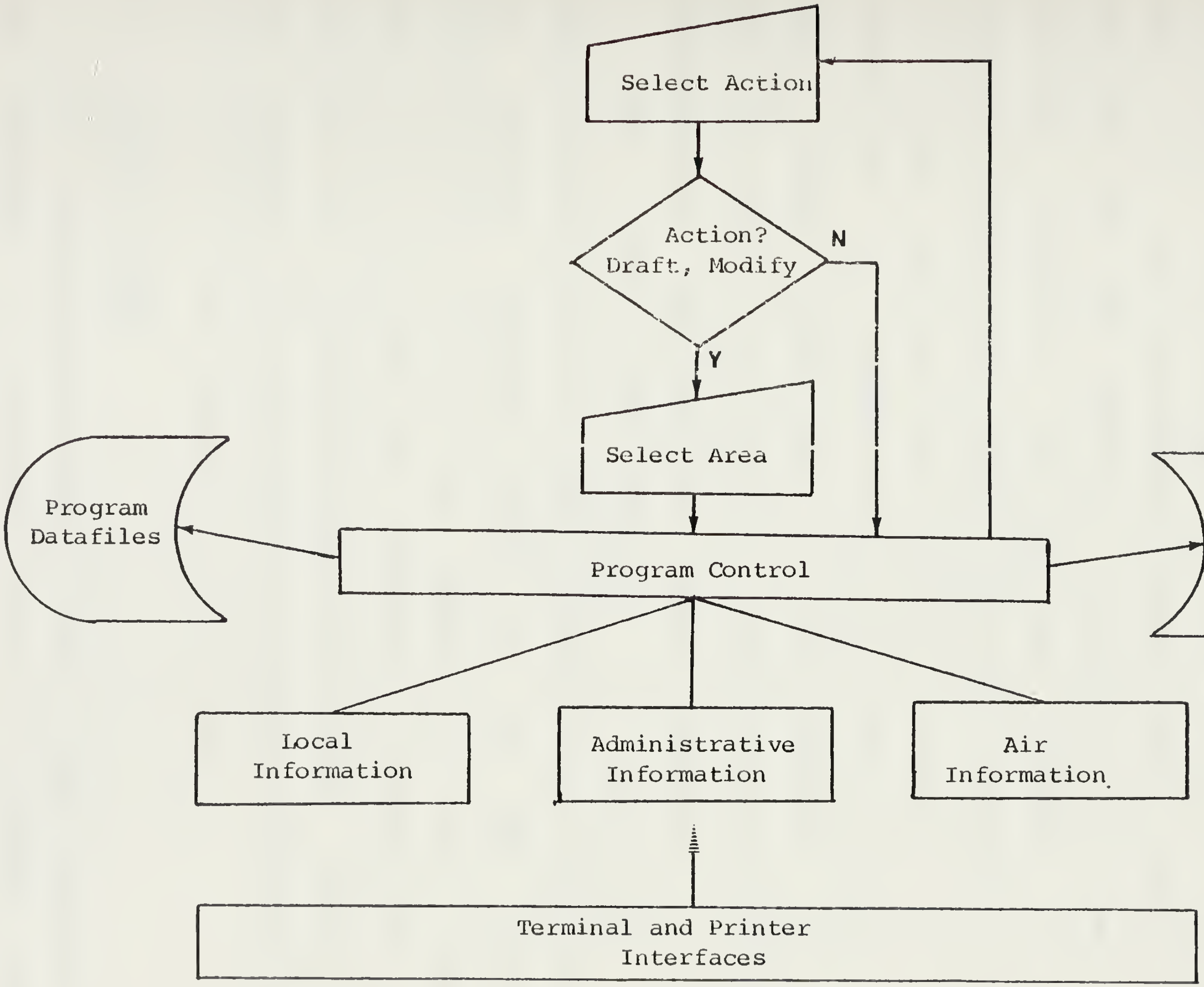


Figure 3-1. ROS UNITREP SUBSYSTEM Overview

(4) Aircraft and Crews Status
(5) Major Equipment Status
(6) Special Capabilities Status
(7) Increased Defense Readiness Status
(8) Reserve Augmentation Status

A chapter of the instruction is devoted to each situation. Details for the particular data sets which comprise the situation and the reporting requirements for each are provided. Review of the data set summaries for each situation reveals that certain data sets are common to all situations. For this reason an additional area, (0) Local Information, was formed to group the common data sets and other message related items. A final situation, (9) Type Commander Reports, was created to separate items which would be designated to be reported by higher authority.

Three areas were selected to demonstrate the implementation approach. LOCAL Information was selected as it provides elements common to any NRS message. ADMINISTRATIVE Status was selected for the relative simplicity of the data sets. AIRCRAFT and CREWS Status was selected for the complex inter-relationships between the data components. The selected situations and supporting data sets/information are detailed in Table 3-1.

B. PROGRAM ACTIONS

Once the logical grouping of data had been confirmed it was necessary to to establish the set of program actions which could be performed on the data.

Table 3-1. UNITREP Data Set/Information - Situation Correspondence

UNITREP SITUATION			
LOCAL	ADMINISTRATIVE	AIR	GENERAL
Message Information	COMMAND	AIRAUTH	AMPN
Addresses	ACTIV	CREWAUTH	DELETE
Originator	MEDIC	AIRSTAT	
Action	REPORG	CREWSTAT	
Info	VERIFY	RECCN	
Date-Time Group			
Priority			
Classification			
OPER			
EXER			
MSGID/UNITID			
POSIT			
RMKS			
DCLAS			

1. ACTION -1- : Provide a Draft Document

The drafting document supports four requirements:

- -- It is the drafter's worksheet and therefore for each data set provides; plain language descriptions of all fields, field length limitations, the source of information (governing directives or pertinent table references) and the data set reporting requirements.
- -- It is the interface between the drafter and the system operator. Information presented conforms directly to the screen displays.
- -- It is the means of reviewing all information as currently reported. Thus each data set or item of information as previously transmitted is reproduced in message format.
- -- It is the means for the releaser to review the report with minimal reference to the governing instuctions. Therefore it contains plain language definitions and and an explanation of any coded items.

Since the draft document has a multiple purpose it is necessary for the system operator to select the purpose for which the document is required. This is accomplished by limiting the selection to forms based on previously transmitted information or forms based on the current information, prior to actual message transmittal. The system operator also selects the situation for which draft information is required since, in general, reports are

prepared for particular situation and do not require the draft information available for other situations.

2. ACTION -2- : Initiate a New Report

This selection allows the program to reset that information which is variable for the individual occurence of a report. Only one UNITREP may be in work at a time. However, in a full ROS including such items as CASREPs, the capability to concurrently prepare several reports should be provided.

3. ACTION -3- : Modify Report Information

This action allows the system operator to change, add or delete information as required when preparing a report. The system operator selects the situation to which the report information applies and program flow is directed to the subprograms for displaying and modifying that data.

4. ACTION -4- : Print Hard Copy Message Format Report

To satisfy a command's archiving requirements, the system provides the capability of producing hard copy output of either the current message (undergoing preparation) or back messages (via either serial or date-time group reference). The information in back messages is not alterable by the system operator. A choice of formats (OCR and fleet broadcast) is allowed.

5. ACTION -5- : Log the Report as Transmitted

In general the date-time group for a message is not assigned until transmittal. When the date-time group is

assigned AND provided to the system, report data is filed for possible future recall, maintaining a serial/date-time group cross reference.

6. ACTION -6- : Erase the Current Report

At times during the report generation cycle (Figure 3-2) a report may be either cancelled or so severely altered that information as currently provided is useless. This action allows the system operator to ´fall-back´ to the period before the report was initially generated.

7. ACTION -7- : Provide Verification Information

The NRS requires the Fleet Commanders in Chief (FLTCINCs) to conduct periodic reviews of all received unit information to insure current and accurate data is being maintained in the WWMCCS database [Ref. 10, para.4.8d]. This action allows all information as currently reported by the unit to be printed for comparison with the FLTCINC feedback verification message.

8. ACTION -8- : Quit

This action allows normal termination of the program, maintaining all information as input during the terminal session.

C. PROGRAM DISPLAYS

Once the system operator has selected to modify information for a particular situation, control of the program passes to the subprogram(s) responsible for allowing

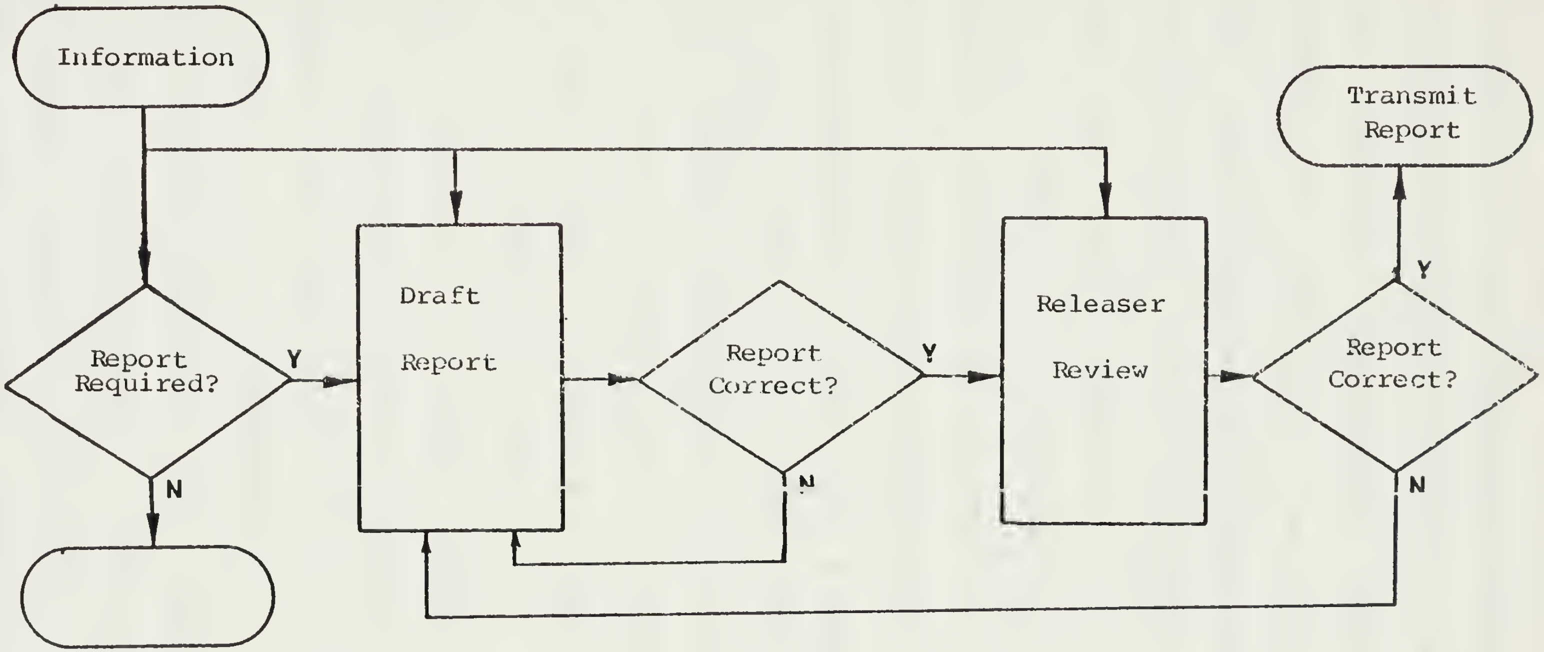


Figure 3-2. Report Generation Cycle

modification for the situation. The displays are grouped in direct correspondence to the drafting document with data set names and fields notated in full form vice abbreviation. All displays are initially menu driven (positioning of the cursor determines the data sets to be selected and inputs allowed) with abbreviated prompts exhibited to notify the operator of legal inputs. Additional HELP information is available for many items by typing a '?' at the appropriate cursor position. Where data field contents are limited to only a few specific entries the choice is by menu selection. Where data field contents are obtained from a table, the table is cross checked to allow only a legal table value. All special purpose fields (e.g. date-time groups, lat/long positions) are checked. All numeric fields are validated and checked for relational comparisons if required. Data text entry in all other fields is limited to the field length as established by the instruction [Ref. 12]. Characters are automatically limited to those allowed for message transmission codes with automatic conversion of lower case letters to upper case. The cursor automatically moves to the next legal position when a selection has been made, the field length has been reached or the operator terminates data entry by a carriage return . The option to reselect items of the current display is provided prior to continuing to the next display. Display formats are provided in Appendix A.

D. DATA STORAGE

All UNITREP information is maintained in a pair of external disk files (UNIT000A, UNIT000B) refered to as the Workfile. This arrangement allows buffering of data by report in the event of hardware, disk or software failure. All modifications are made to the workfile. When a report is being logged as transmitted (no further update for that serial message is required), the Workfile is copied to a separate UNITREP files storage disk inserted in the microcomputer B: disk drive. The filenames UNITxxxA and UNITxxxB, where xxx represents the report serial number, are used. This allows the data to be recalled for verification or message copy purposes. The Workfile is reset as required when a new report is initiated.

IV. IMPLEMENTATION

This chapter discusses the author's ROS UNITREP SUBSYSTEM implementation in the Ada language. A review of the limitations of the JANUS/ADA subset, a program overview and a discussion of the the various components are provided. The ROS UNITREP Operator's Manual (Appendix A) and the ROS UNITREP COMPUTER LISTING (Appendix C) may be refered to for more detailed information.

A. ADA SUBSET RESTRICTIONS

The JANUS subset of the Ada language provides many of the sophisticated features required in a high level language. Data abstraction and strong data typing allows a clearly readable coding style with the ability to uncover and correct errors during compilation vice run time. The Ada package concept allows a modularization that leads to localized and easily modifiable code which can be altered without serious consequence to the program as a whole.

However, the lofty goals of the Ada language do not easily lend themselves to implementation on small scale microcomputers. Quoting from the JANUS/ADA Package User Manual:

> "The major design tradeoffs in Janus were power versus simplicity. A major effort was made to keep the Janus subset as small as reasonable. This was done in order to complement yet not inhibit the ambitious nature of Ada.

> Janus was designed specifically for the microcomputer environment. Therefore, every Ada construct which would have required excessive machine resources was rejected. The complicated features of Ada that are viable only for large computers, e.g. parallel processing, are not included in Janus." [Ref. 9: p.1-1]

It is not the intent to review all JANUS deficiencies here. A complete list is available in the JANUS/ADA PACKAGE USER MANUAL [Ref. 9: App. L]. Those items which detracted from the ROS UNITREP implementation will be discussed.

1. Lack of a Standard I/O (Input/Output) Package

JANUS implements this concept through the use of library modules. Four of these library modules were used by the program. The subprograms which are provided by these modules are listed in Appendix B. Unfortunately many Ada I/O subprograms were not implemented, particularly those addressing full screen console operation and printer usage. This required creation of the necessary functions.

2. Lack of Aggregates

Aggregates are written forms denoting a composite value, essentially being record and array literals. Without aggregates all composite variables required initialization at run time in the package body. This detracts from localizing such items as constant strings with their value at the declaration.

3. Type Limitations

The JANUS compiler maintains internal tables for certain objects encountered during compilation. A maximum of

ninety (90) different data types may be specified. As all UNITREP data sets and many of their components were represented as separate types, the limit was exceeded. This necessitated modification of the program and data structure into modules where the total number of types, for each module, were within limits.

4. Lack of Exception Handling

This Ada feature allows a program to react to an event which causes suspension of normal program operation. Without the built-in exception handling capability all keyboard input was character and string buffered and the content interpeted in separate subprograms. Subtyping could not be usefully employed since this constraint_error type of exception resulted in program aborts.

5. Lack of an Overlay Capability

In the microcomputer environment the utilization of overlays for medium and large programs is a necessity. The lack of overlays required physically dividing the program into four independent executable files and introduced the need for communication and control components. Since the program is structured with a large set of supporting subprograms (24 kilobytes) this information must be reproduced for each module. Therefore 72 kilobytes of secondary storage are duplicated code. Additional disk access time is required to load the larger program modules in comparison to what would be much smaller overlays.

6. Lack of Random Access (Read/Write) External Files

In JANUS external files can only be read or written. Such restrictions require the entire external file to be loaded into main memory for the modification of a single component, even if the file is static in composition. For large external files the impact on main memory and program size is prohibitive. This also prevents the use of dynamic files, which resulted in the inability to provide the user with an on line file update capability for maintaining certain tabularized data.

7. Implementation of the Type String

JANUS uses dynamic strings and supports full string operations. Although not detrimental for the purposes of this program, portability and compatability problems may occur with other Ada compilers.

B. PROGRAM OVERVIEW

Prior to reviewing the program, an explanation of terms used in the discussion is necessary:

-- subprogram: a procedure or function.

-- package: the Ada construct for the logical grouping of information.

-- package specification: that part of a package which contains the declarations of types, constants, variables and visible subprograms.

-- package body: that part of a package which contains the information necessary to support the subprograms declared in the package specification.

-- module: the executable code (.COM disk file) which results when packages are compiled and linked. The module name is that of the highest level package.

The lowest level in the program hierarchy consists of the basic program support features which provide console and printer communication, global information, program module communication and utility subprograms. This level consists of the PACKAGEs consio,printio,urglobal and urutil as shown in Figure 4-1.

The UNITREP data structure definition is divided into two components by PACKAGEs unitrepA and unitrepB. The respective subcomponents (PACKAGEs urgltbl, urlocalA, uradminB and urairB) and structure are shown in Figure 4-2.

PACKAGE unitrep provides the main selection subprograms for choosing and processing the desired program actions and areas, as well as the constructs for directing program flow to the other control packages (unitrep1, unitrep2, unitrep3). PACKAGE unitrep1 also interfaces to the LOCAL Information subprograms contained in PACKAGEs urlocal and urlocal1. Modules unitrep and unitrep1 operate on data as defined by unitrepA. File checking and handling subprograms for the 'A' structure are contained in PACKAGE filerA. The PACKAGE initialA is used to set the contents of the external disk workfile (UNIT000A) as required for initial program operation. (Figure 4-3)

PACKAGE unitrep2 controls access to all ADMINISTRATIVE processing subprograms (PACKAGE uradmin) as well as all AIR

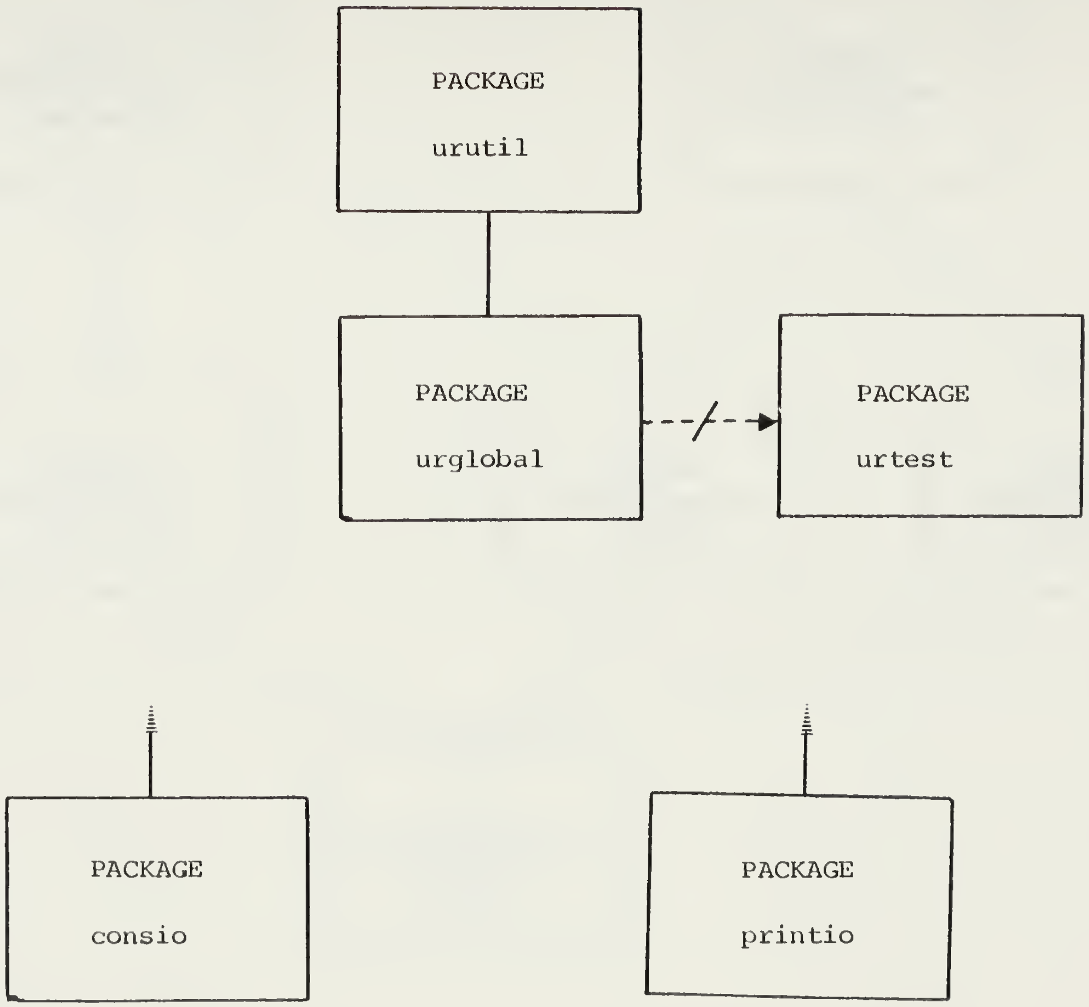


Figure 4-1. Basic Support Organization

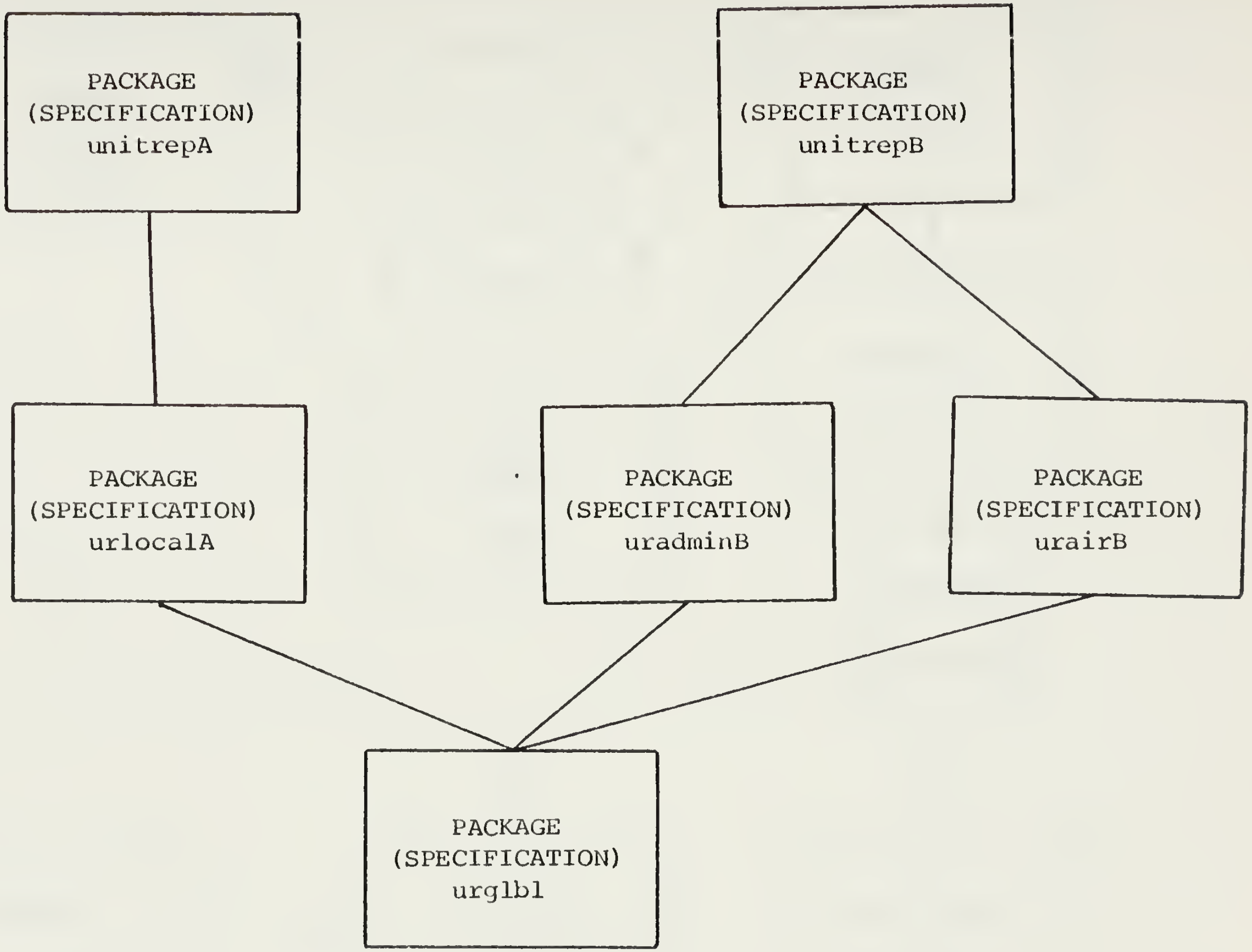


Figure 4-2. Data Structure Organization

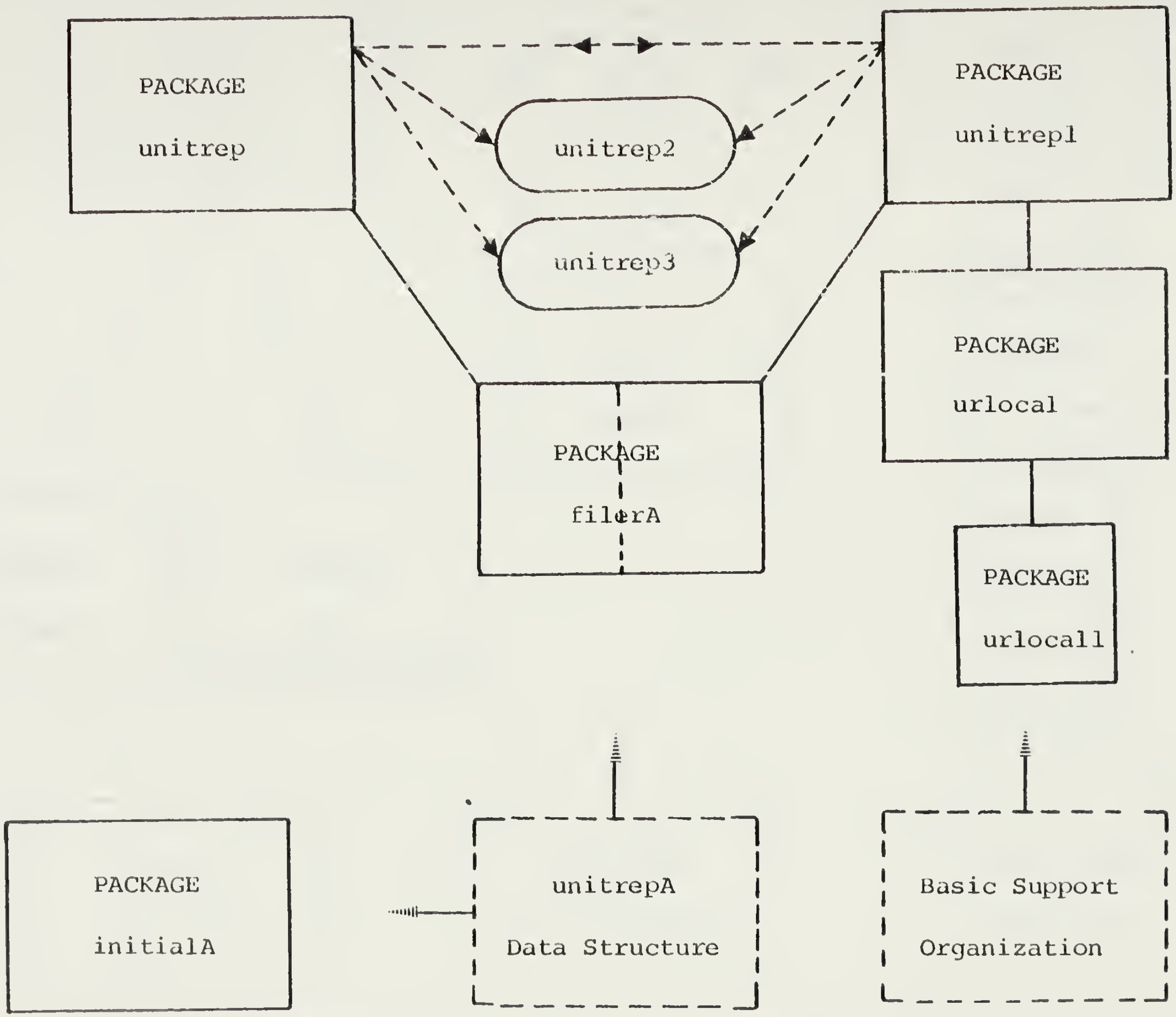


Figure 4-3. Program Structure for 'A' Data

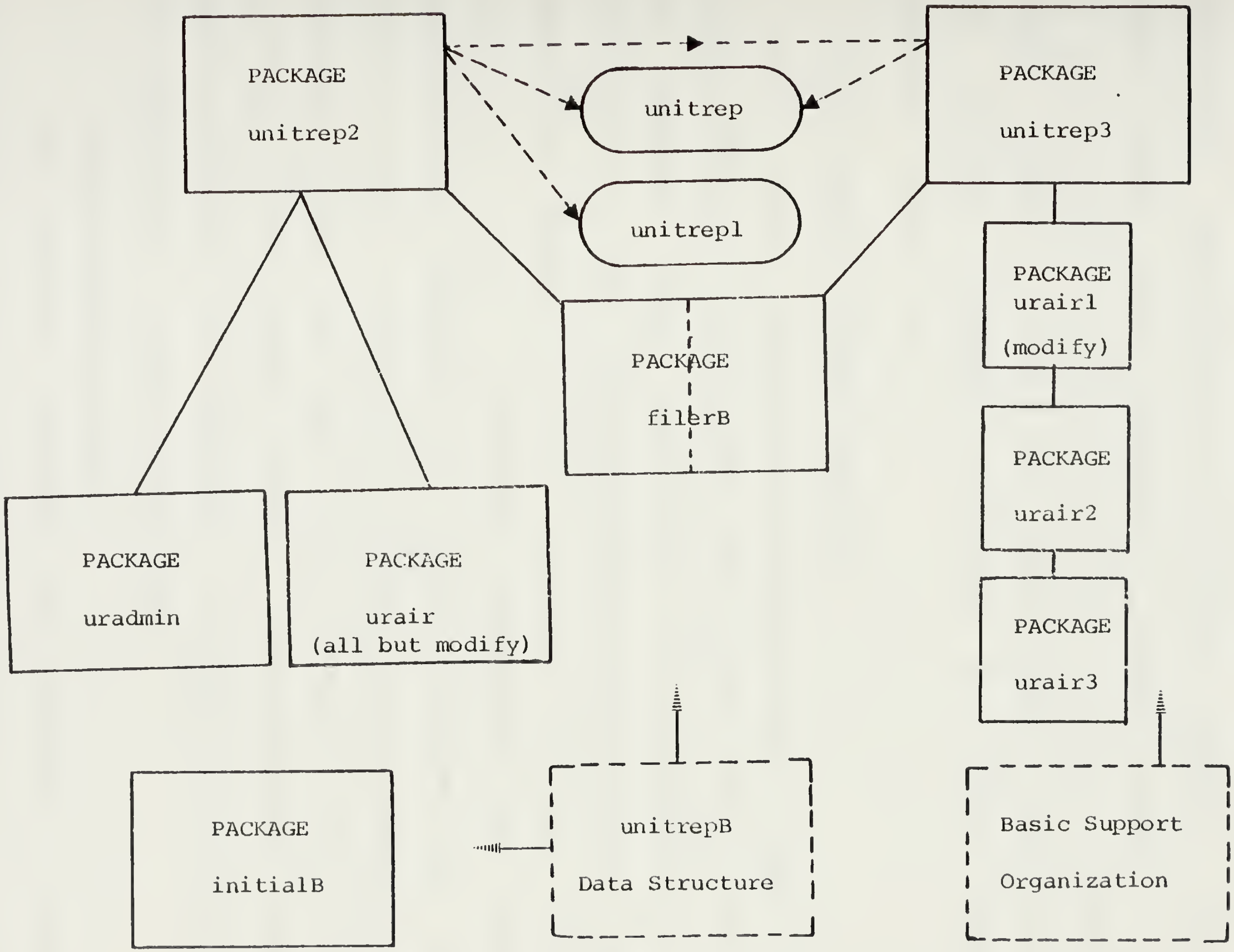


Figure 4-4. Program Structure for 'B' Data

processing subprograms (PACKAGE urair) except those concerning the modification of AIR information. PACKAGE unitrep3 is the interface to the AIR information modification subprograms as contained in PACKAGEs urair1, urair2, urair3. Modules unitrep2 and unitrep3 operate on the 'B' data structure. File operations are contained in PACKAGE filerB. PACKAGE initialB sets the contents of the external disk workfile (UNIT00B) as required for intial program operation. (Figure 4-4)

C. BASIC PROGRAM SUPPORT COMPONENTS

Four common packages are required by each of the four program modules to provide the basic support items. Two of the packages - consio and printio - are designed to be customized to the specific hardware configuration of the terminal and printer. A third - urglobal - provides global information, user assistance subprograms and module communication information to control program flow. The fourth - urutil - consists of routines for the checking of redundant fields and higher level, general subprograms. Initially a fifth package - urtest - was included to gather testing information on the system. Although not included in the final product it will be discussed.

1. Console Package (consio)

This package satisfies three purposes: terminal customization, keyboard character conversion and screen

oriented (x/y) addressing procedures for data input and output.

a. Terminal Customization

In general, all terminals have a specific interpetation of the American Standard Code for Information Interchange (ASCII) character codes in relation to terminal functioning (e.g. cursor positioning, screen backspace, clear screen). Any program function or procedure dependent upon a specific interpetation of one or more of these codes would require modification for the specific terminal. This would be unsatisfactory in light of the number of different terminals available and the number of different subprograms used to support the full screen, menu driven user interface. For this reason items peculiar to terminal functioning are hidden from the program in the consio package body. For localization purposes those ASCII character codes which have accepted meanings (carriage return, bell, horizontal tab) are declared in the package specification, while non-standard interpetations are declared in the package body. When customizing to a particular terminal only those items in the package body must be modified. The package body is then recompiled (vice recompiling the entire program) and the four modules are relinked. An example of the differences is between two terminals is given in Table 4-1. Two procedures directly relate to terminal customization:

Table 4-1. Terminal Function - ASCII Character Code Correspondence.

TERMINAL FUNCTION	ASCII DECIMAL CHARACTER CODE Datamedia Elite 2500	Micro-Term ACT-V
(In Specification)		
Carriage Return	13	13
Horizontal Tab	9	9
Bell	7	7
(In Body)		
Rubout (delete)	127	127
Screen Backspace	8	8
Clear Screen	30	12
Position Cursor	12	20

-- PROCEDURE gotoxy(x,y: IN INTEGER);

The program is coded on the assumption that it will be implemented on a terminal allowing at least 23 rows (numbered 0-22) and 79 columns (numbered 0-78). This meets the standard 24 row by 80 column display with a small safety margin. This procedure modifies the linear representation used in the program (x - column, y - row) to that set of character codes required by the terminal to position the cursor. The conversion algorithm to the correct value must be modified for the specific terminal, the consio package body recompiled and the four control packages (unitrep, unitrep1, unitrep2, unitrep3) relinked prior to distribution and use.

-- PROCEDURE clear_screen;

Sends the clear and home character to the terminal. Direct modification of this procedure is not neccessary since the clear screen character code is declared at the beginning of the consio package body.

b. Character Conversion

The output of the program is a message for ready for transmission. Therefore, only those characters which are compatible with both the Optical Character Reader (OCR) and five-level (BAUDOT) character codes are acceptable. Since most keyboards provide the 128 ASCII characters, some means of limiting to these character sets is necessary.

```
-- PROCEDURE baudot_convert(char: IN OUT CHARACTER);
```

This procedure is hidden from the program in the consio package body and may be modified to any desired character set. No action is taken on control characters (ASCII Decimal 0-31) with the exception of ASCII 3, Control-C, which is interpeted to be a user commanded program abort, and ASCII 9, tab, which is converted to an 'X' character to allow the tab key to be used in menu selection functions. All lower case letters are automatically converted to upper case. The set of all allowable characters is:

0 1 2 3 4 5 6 7 8 9 A B C D E F G H I J K L M N O P
Q R S T U V W X Y Z - ? : $ & # ' ( ) . , ; / "
space

Any other character input from the keyboard is automatically converted to ASCII 0 and ignored in processing. The operator is notified by an aural tone that the depressed key is unacceptable.

c. Screen Oriented Input-Output Procedures

The decision to use a full screen format and menu selection required a complete range of screen oriented subprograms. The majority of the subprograms in this class can be identified by the 'xy' in the subprogram name.

```
-- PROCEDURE getxy_immediate(x,y: IN INTEGER;
                             char: OUT CHARACTER);
```

This procedure is used to character buffer all data input and during menu selection. The cursor is positioned to

the indicated coordinates (PROCEDURE gotoxy) and exactly one keyboard input is read and converted to an acceptable character, ´char´ (PROCEDURE baudot_convert).

```
-- PROCEDURE getxy(x,y: IN INTEGER;
                   temp: OUT STRING;
                   field_length: IN INTEGER);
```

This procedure builds a string, ´temp´, character by character (PROCEDURE getxy_immediate), operating on each character as received. All ASCII 0 characters are ignored and on screen backspacing and deletion are allowed. The size of the string is automatically limited to ´field_length´ characters. Keyboard input is terminated when receiving a carriage return or upon reaching the field length. The string is then stripped of any trailing blanks which may have been entered. The string is then returned to the calling subprogram for use.

```
-- FUNCTION mark(x,y: INTEGER) RETURN BOOLEAN;
```

This function is used in menu selection areas where a boolean value is required by the program. The function accepts a keyboard input (PROCEDURE getxy_immediate) and determines if it meets the requirements used to indicate a mark. TRUE is returned if an ´X´ or tab key has been depressed, FALSE is returned if the carriage return or space bar have been depressed. If one of these four keys was not selected the input is ignored and the user notified by an aural tone to reselect.

```
-- PROCEDURE putxy(x,y: IN INTEGER;
                   temp: IN STRING);
```

This procedure puts the contents of the input variable 'temp' at the indicated coordinates (PROCEDURE gotoxy).

```
-- PROCEDURE clear_field(x,y,len: IN INTEGER);
```

This procedure is used to erase the entire contents of a data field at the indicated coordinates (PROCEDURE gotoxy). This applies only to the terminal display, not the actual data field variable.

2. Printer Package (printio)

The printer package supports the program customization to a particular printer and the setting of several printed copy formats.

a. Printer Customization

Since a continuous page printer was used, only printer enabling/disabling and page ejection mechanisms were required to be constructed.

```
-- PROCEDURE printer_on; PROCEDURE printer_off;
```

JANUS/ADA communicates through the standard CP/M-80 filenames for input/ouput devices. The measures described in the manual using the LST: device were followed [REF. 9: p. 2-5].

```
-- PROCEDURE new_page;
```

Although a standard ADA construct, this procedure was not available in the JANUS/ADA subset. It was simulated by putting the form-feed character (ASCII Decimal 12) to the

CP/M LST: device. The page_eject constant is hidden in the printio package body to be easily modifiable.

b. Output Formats

Three output formats were selected for demonstration purposes: message (msg), which simulates the format received during fleet broadcasts; Optical Character Reader (ocr), which is used for message preparation; text (text), which is used to prepare printed data not message oriented.

```
-- TYPE  format IS (msg,ocr,text)
   PROCEDURE  set_format(format_type:  IN format)
```

The various parameters for each format are listed in Table 4-2. Formats are set either by default or, where applicable, by the operator through menu selection options. The procedure controls the actual parameters and is hidden within the printio package body. It is adjustable to any format by variation of the format parameters.

```
-- PROCEDURE put_printer(content: IN STRING);
```

Once the format is selected all printer directed output is passed throgh this procedure. In addition to writing to the printer the proper format is maintained by adjusting header lines, line spacing, margins and page ejects. The line count is adjusted at each call to new_page.

3. Global Package (urglobal)

The global package provides the declarations of global types, constants and variables, the program status

Table 4-2. Format Parameter Specifications

		FORMAT	
PARAMETERS	Message	OCR	Text
max_lines_in_page	42	19	55
max_line_length	69	69	80
line_spacing	1	2	1
header_lines	9	6	6
margin (spaces)	5	5	0
end_trans (symbol)	"BT"	"NNNN"	N/A

structure for program module communication, general use subprograms and user assistance subprograms.

a. Global Declarations

The package specification details the types, constants and variables which may be accessed by all higher levels of the program. Common use temporary variable identifiers for the specific types INTEGER, STRING, CHARACTER, BOOLEAN and FILE are listed here to maintain uniformity of reference throughout the program. Constant characters are used in case statements and equality comparisions - letters are recognized by the lower case representation of the character (e.g. ′a′ for ′A′) while other characters are represented by plain english (e.g. ′space′ for ′ ′). Constant strings (of length one) are recognized by the ′_mark′ suffix (e.g. ′a_mark′ for "A") or plain english (e.g. ′dash′ for "-"). String constants are initialized in the package body as JANUS/ADA does not allow the use of aggregates in the package specification.

The eight designated program actions are carried as both a type (′action′: draft, neww, print, etc.) and an array of booleans (′action_is′) since JANUS/ADA does not provide for the writing to storage of enumeration types (other than boolean). The ten program areas, consisting of the eight UNITREP situations, the LOCAL Information area and the TYPE COMMANDER report section are similarly represented (′area′: local, air, admin, etc.;′area_is′).

b. Program Module Communication

Since the program is structured in multiple modules, some information passing is necessary to maintain program control without requiring redundant user interaction. This information is contained within the variable 'program' whose structure is the type 'program_status'. An explanation of each component of program follows.

-- unit: TYPE unit_type IS
(ship,submarine,air_unit,shore,other)

This item is used to distinguish the reporting requirements of certain data sets (e.g. only ships and submarines are REQUIRED to report their positions (POSIT set), only ships and submarines are allowed to report the physician status (MEDIC set)). This allows the program internally to required or disregard these fields. 'Unit' is set in PACKAGE urlocal, PROCEDURE process_local_infrequent.

-- initial_entry: BOOLEAN

This item enables the Sign On display screen. It is reset in PACKAGE unitrep during the initial execution of the main program and set when the user elects to quit the system in PACKAGE unitrep, PROCEDURE process_action.

-- print_trailer: BOOLEAN

This item is used by PACKAGE urlocal, PROCEDURE print_msg, to determine if message trailing items (RMKS

and ICLAS data sets) are to be printed. This sequences the proper printing order. It is set in PACKAGE unitrep after sign on and reset in PACKAGE unitrep when action -8- (quit) is selected.

-- message_transmitted: BOOLEAN

This item is used by PACKAGE urair, FUNCTION print_with_delete to determine when a deleted data set is required to be printed (Figure 4-5). It is set in PACKAGE unitrep, PROCEDURE process_action when a message is logged as transmitted (action -5-).

-- current_action: action_state

This item communicates the proper action to each program module. It is set in PACKAGE unitrep, Procedure choose_action.

-- current_area: area_state

This item is used by the various control packages to direct program flow only to the user selected areas. It is set in PACKAGE unitrep, PROCEDURE choose_areas.

-- current_format: format_state

This item is set either by default of through user selection in PACKAGE unitrep, PROCEDURE choose_action, and maintains the proper format during a print cycle.

-- printing_line: INTEGER

This item is set by the variable line_count in PACKAGE printio. It maintains the value of the line currently

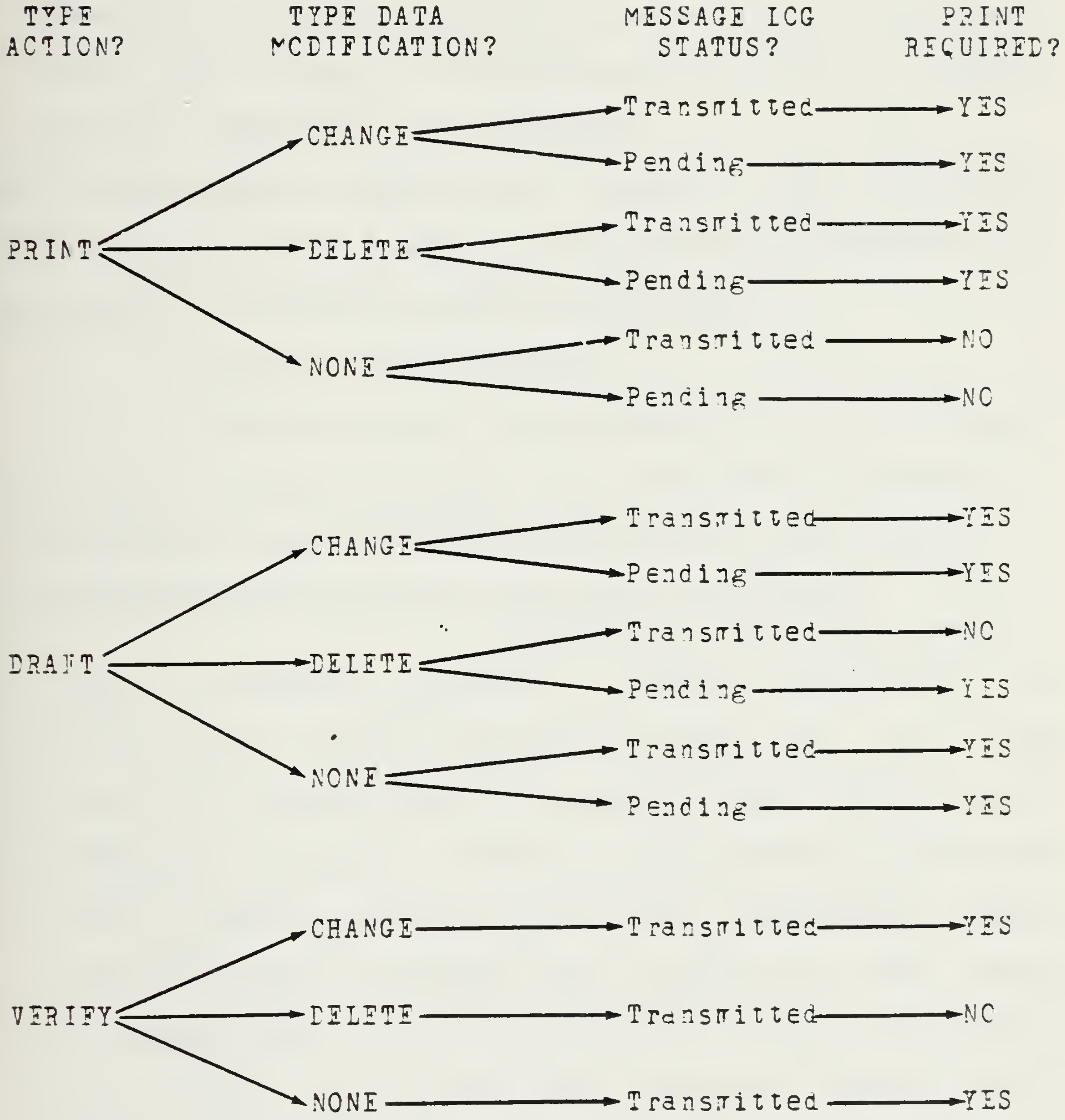


Figure 4-5. Data Set Print Decision Tree

being printed when switching between modules during any action which requires printed output.

-- workfile: STRING(14)

This item holds the name of the current data structure on which the program is operating and is set in PACKAGE unitrep, PROCEDURE process_action.

The program status information is stored at the termination of any module and is loaded at the beginning of execution of any module (see PROGRAM CONTROL COMPONENTS).

c. General Use Subprograms

Subprograms in this grouping are used throughout the program to support other, more complicated, constructs.

-- PROCEDURE border; provides the display screen outline

```
-- PROCEDURE process_comment(data_set: IN STRING;
                             comment: IN OUT comment_set);
```

This procedure provides screen processing of amplification (AMPN) and remarks (RMKS) data sets. The content of ´comment_set´ is described under PROGRAM DATA COMPONENTS (urglbl). Comments are processed as external files in order to reduce data storage requirements. These free text files are purged once a message has been logged as transmitted. If a file has been created previously, the contents are loaded and displayed (Figure A-5). Otherwise a new file for that data set is created and the user enters the free text content. When the user has completed the initial entry of data, noted by reaching

the maximum allowed comment lines (10) or inputing a carriage return on a blank line, the option is presented to line edit, erase or continue with the program. Once the user is satisfied with the free text the program stores the content in a file on the A: disk with a filename of 'data_set' and a filetype of ".AMP" (e.g. "POSIT.AMP"). Remarks are processed as 10 pages of comments to allow 100 total lines. In situations where multiple data sets are possible the variable 'data_set' is modified at the point of the procedure call to indicate the instance of the 'data_set' with integer suffixes (e.g. CREWSTAT amplification, 'data_set' := "CRWSTA12" where '1' indicates the first aircraft type and '2' indicates at the second location; AIRAUTH amplification, 'data_set' := "AIRATH1" where the '1' indicates the first aircraft type).

-- PROCEDURE print_comment(data_set: IN STRING);

This procedure is used to print the content of amplification sets only. The external file, 'data_set'.AMP, is located and "AMPN/" is appended at the beginning of the first line. All lines are printed and the end of data delimiter ,"//", appended to the last line. Although remarks are processed as multiple page amplification sets, a special procedure must be used for printing as the placement of the delimeter is variable (see PACKAGE urlocal, PROCEDURE print_msg).

-- FUNCTION required_print(changed_item: BOOLEAN)
RETURN BOOLEAN

Based on data set printing requirements (Figure 4-5), this function determines if a non-deletable data set is to be printed.

d. User Assistance Subprograms

The user assistance subprograms notify the system operator of input errors, provide additional HELP information or provide draft document information.

-- PROCEDURE error(number: IN INTEGER);

In certain situations of illegal data entry or where notification to the user of certain actions required to be followed are necessary, error messages are used. The error number is set at the procedure call and the external ERRMSG file is accessed. The required ´number´ of lines in the file are searched and the appropriate error message is dislayed at the bottom of the current screen display with an aural tone. The procedure waits for a user input before clearing the error message. Error messages are contained in Table 4-3.

-- PROCEDURE help(data_set: IN STRING);

This procedure allows the user to obtain upto seventeen lines of information pertaining to the particular item at the current cursor position. When the user enters a question mark (´?´) at a valid location the external HELP.TXT file is accessed and a search made for

Table 4-3. Error Messages

```
 1: DateTime Group Incorrect
 2: The required file, UNITxxx, is not on the B: disk
 3: The workfile is empty - Copy last UNITxxx to UNIT000
 4: The Pending UNITREP has not been logged as transmitted
 5: Select the LOG PENDING UNITREP AS TRANSMITTED Action
 6: Incorrect Input - Reselect
 7: Pending UNITREP has not been Transmitted. Select 3,5,6
 8: The Current Workfile has been transmitted - Initiate New
 9: Submitted Datetime Group does not cross to a serial
10: Cannot Log an already transmitted UNITREP
11: Invalid serial number
12: -----------------not used
13: Current Workfile has been erased
14: Require <X>/tab or <ENTER>/space-bar as the input
15: Activity Code not found in Table B-6
16: Must choose at least one
17: Amplification Comments erased
18: Invalid LAT/LONG Position
19: -----------------not used
20: Originator Address is a mandatory item
21: Required information for a classified message
```

Table 4-3. Error Messages (cont'd)

```
22: Unit Identification is mandatory
23: *** WARNING *** last serial should only be changed ....
24: ...on initialization -- RESELECT if still desired
25: <ENTER> to continue, ENTER <X> to reselect:
26: Remarks erased
27: Cannot amplify a set which has not been changed
28: <X>¦tab, <ENTER>¦space, <A> only keys allowed
29: Amplification not allowed for this set
30: Maximum of 15 addressees allowed
31: Set has been DELETED - cannot modify - RESTORE first
32: An Aircraft Type is required to continue
33: Aircraft Type not found in Table B-2
34: This set has not been deleted - cannot RESTORE
35: Once a set has been modified it cannot be deleted
36: Invalid Location
37: HOME PLATE is a synonym for present location
38: Possessed aircraft must equal FMC + PMC + NMC
39: Crews Formed cannot be less than Crews Ready
40: Delete individual capability before adding another
41: Reconnaisance Capability not found in Table B-3
42: Invalid Aircraft Type
```

'data_set'. Once found the contents of the file are displayed until the delimiter (a single period on a line) is reached. The procedure terminates when the user makes a keyboard entry and control is returned to the calling subprogram at a point allowing use of the information viewed.

-- PROCEDURE view_table(filename: IN STRING);

This procedure is used in conjunction with the help subprogram when the data field item under enquiry is contained in a table supplement to the instruction for which the program is designed [Ref. 10: Tables B-1 thru B-14]. The procedure notifies the user that that table entries are available, by message and an aural tone, following a HELP request. The user may elect to view or not view the table. If the decision is to view, the table entries are displayed seventeen per screen. The user can exit viewing at any point or will automatically exit at the end of the table. Program control returns at a point allowing use of the information.

-- PROCEDURE draft_aid(data_set: IN STRING);

This procedure provides the user hard copy information to assist in the drafting of a report. The information is stored in an external file, DRAFT.TXT, and is automatically accessed during draft document preparation. The DRAFT.TXT file is searched for 'data_set' and all contents until the delimiter (a single period on a line)

are printed. A sample draft document prepared using the content of DRAFT.TXT is contained in Appendix A.

4. Utility Package (urutil)

The utility package provides several subprograms which are also of general use but which were separated from PACKAGE urglobal for their similarities.

```
-- PROCEDURE verify_number(number_of_digits: IN INTEGER;
                            temp: IN OUT STRING;
                            success: OUT BOOLEAN;
                            number: OUT INTEGER);
```

The necessity of string buffering the input (as JANUS/ADA has no exception handling capability) required a capability to determine if certain string inputs were numeric in nature and if so to provide the integer value. This procedure checks all characters of the string 'temp', insures they are numeric and then converts the string to an integer value which is returned in the variable 'number'. If 'temp' does not represent a number or if the number is not positive and in range (0 to (10**number_of_digits)-1) then 'success' is set FALSE. Leading spaces within the string are converted to the zero ('0') character.

```
-- FUNCTION getxy_digits(x,y,number_of_digits: INTEGER)
                                         RETURN STRING;
```

This function is used where the integer value of the number is not immediately required and where x/y addressing is prefered. The cursor is positioned at x,y

and a maximium of 'number_of_digits' characters are accepted. Input is terminated on a carriage return of when reaching 'number_of_digits'. Any trailing blanks are stripped and the string is sent to PROCEDURE verify_number for processing. If the input is an invalid number the user is notified with an error message and the sequence is repeated within the function. The function terminates when a set of numeric characters which meet the various tests is processed. That set of numeric characters is then returned as the function value.

```
-- PROCEDURE verify_dtg(dtg: IN OUT STRING;
                        success: OUT BOOLEAN);
```

Error checks 'dtg' for all legal date-time groups. If an invalid date-time group is received the user is notified by an error message and 'success' is set to FALSE. The date-time group template, "ddhhmmZMMMyy" is returned after the user clears the error message.

```
-- PROCEDURE verify_lat_long(lat_long: IN OUT STRING;
                             success: OUT BOOLEAN);
```

This procedure error checks latitude/longitude position reports. If the entered 'lat_long' is incorrect the user is notified by an error message and 'success' is set to FALSE. The position template, "ddmmDC-dddmmDC" is returned when the error message has been cleared.

```
-- FUNCTION checksum(temp: STRING) RETURN INTEGER;
```

This function calculates the checksum for a string of numeric digits and returns that integer value.

```
-- FUNCTION zero_pad(number_of_digits: INTEGER;
                     temp: STRING) RETURN STRING;
```

This function adds character zeroes ('0') to the beginning of the string 'temp' if required to fill the string out to 'number_of_digits'. It is used to maintain numerical string variables of consistent field widths.

5. Test Package (urtest)

This package was initially envisioned as a means of automatically collecting test data on program operation. This includes such items as user identification, sign on and sign off times, program recognized error tabulations, the number of help accesses required and a method of gathering user comments. Use of this package was discontinued because the lack of JANUS/ADA random access read/write file procedures required separate external test data files to be generated each time a program module was entered. This produced a number of files which frequently exceeded directory space during program operation causing program abort. Elimination of this package increased the space available in main memory and allowed the use of a larger file buffer size which increased file handling capabilities. References to the test procedures remain in the program listing and are recognized by the characters "--#" which effectively comment out all calls to the applicable subprograms.

D. PROGRAM DATA COMPONENTS

The program uses nested Ada record types to define the data structure. At the lowest level the data is structured according to the data set definitions [Ref. 10: pp. 11-5 thru 11-47]. The type identifier consists of the data set name and a '_set' suffix (e.g. 'operation_set', 'medic_set'). Each component of this record represents a data field of the data set. Data sets which are conditional (i.e. reported only when necessary) are identified by a boolean flag ('change'). Data sets which may be deleted are identified by an additional boolean flag ('delete'). If a data set can be amplified the record contains the component 'ampn' which is defined in PACKAGE urglbl.

These data sets are then grouped in another record according to the UNITREP situation to which they apply (e.g. 'local_set', 'air_set') by the appropriate package (urlocalA, uradminB, urairB). These composite area sets are finally grouped in the 'unitrep_set' record as declared in PACKAGEs unitrepA and unitrepB. The identifier 'u' is used in both packages to denote the unitrep data variable. This allows the multiple data structures to be collapsed to a single data structure for Ada subsets without the JANUS/ADA type limitation.

1. Free Text Data (urglbl)

The NRS instructions allow the majority of NRS data sets to be amplified by using the free text set 'AMPN'. An

amplification set consists of upto 10 lines of text, 69 characters per line. In order to limit the printed line length in a message to 69, the program constrains the free text line length to 62 characters (data set identifiers and delimiters account for the other 7 characters). PACKAGE urglbl consists of the declarations necessary to support the 'AMPN' data set. As the actual text is stored in an external file (see PACKAGE urglobal, procedure process_comment) the type 'comment_set', consists of only a flag ('change') and a count of the 'number_of_lines' in the set. This count is used when processing remarks sets (which are interpeted as upto 10 pages of comments.

2. Local Information Data (urlocalA)

This package defines those data items necessary for the construction of any NRS/UNITREP message. Data sets included in 'local_set' are generally noted as mandatory by the instruction. In addition to the data sets, type 'message_set' contains information relevant to the particular message (e.g. date-time group, precedence, classification, declassification). The type 'status_set' maintains the items used in the message identifier line (MSGID or UNITID).

3. Administrative Information Data (uradminB)

PACKAGE uradminB contains the declarations for the types used in preparation of administrative information

[Ref. 10: pp.4-2,4-3]. These sets are relatively simple as no deletions or complex relationships are involved.

4. Air Information Data (urairB)

Aircraft and Crews Status information [Ref. 10: pp. 6-6,6-7,6-8] requires a much more complicated structure. The user may submit a variable number of these data sets depending on the type aircraft and also the location of the aircraft. Because of the static nature of the data structure several design restrictions were required. In examining the air data sets it is clear that the aircraft type is a requirement of all. Therefore a set number of aircraft types is required. An arbitrary value of four (4) was selected. Three data sets (AIRSTAT, CREWSTAT, RECCN) require location information. Again arbitrarily a value of four (4) was selected. The location information is a subcomponent of the aircraft type, therefore up to sixteen (16) total locations are allowed. The structure and size of the air_set record is more clearly displayed in Table 4-4. The size of the data structure became relevant because of the the storage required in main memory during program execution. A rough formula for determining the size of the air data structure is:

```
 size  = 1.2 x (3 + 'max_ac_types' x
                             (27 + 'max_locations' x 149) )
```

'Max_ac_types' and 'max_locations' are declared as constants in PACKAGE urairB.

Table 4-4. Air Data Storage Requirements

DATA ITEM	RAW BYTES OF STORAGE REQUIRED		
TYPE air_set IS RECORD			(2495)
change:			1
number_of_ac_types:			2
ac_type: ARRAY(1..max_ac_types) OF			4 x 623
TYPE aircraft_type_set IS RECORD		(623)	
change:		1	
delete:		1	
iss:		7	
ac_auth: TYPE auth_set IS RECORD		8	
change:	1		
delete:	1		
iss:	2		
ampn:	3		
crews_auth: TYPE auth_set IS RECORD		8	
change:	1		
delete:	1		
iss:	2		
ampn:	3		
number_of_locations:		2	

Table 4-4. Air Data Storage Requirements (cont'd)

DATA ITEM	RAW BYTES OF STORAGE REQUIRED				
location: ARRAY(1..max_locations) OF					4 x 149
TYPE dispersed_aircraft IS RECORD				(149)	
change:			1		
delete:			1		
iss:			30		
airstat: TYPF airstat_set IS RECORD			13		
change:		1			
delete:		1			
possessed:		2			
fmc:		2			
pmc:		2			
nmc:		2			
ampn:		3			
crewstat: TYPE crewstat_set IS RECORT			9		
change:		1			
delete:		1			
formed:		2			
ready:		2			
ampn:		3			
recon: TYPE recon_set IS RECORD			95		
change:		1			
delete:		1			
number_of_capabilities:		2			
capability: ARRAY (1..max_recon) OF		11 x 8			
STRING(8):	8				
ampn:		3			

5. Composite Data (unitrepA, unitrepB)

PACKAGE unitrepA consists of only the LOCAL structure define in PACKAGE urlocalA. PACKAGE unitrepB consists of both the ADMINISTRATIVE and AIR structures from PACKAGEs uradminB and urairB respectively.

E. PROGRAM CONTROL COMPONENTS

Program flow is directed according to the actions and areas which are selected by the operator. Without an overlay capability, as the program size expanded it was necessary to physically separate executable portions of the program to remain within main memory space limitations (64K). The program control components are responsible for operator selection of action, selection of area(s) when required, direction of program flow to the necessary modules and the data file manipulations required for support of the selected action/area(s).

1. Primary Control (unitrep)

PACKAGE unitrep provides the program initial (sign on) operations, the selection of program actions, the necessary contructs for supporting the selected action and the selection of area(s) when required.

a. Program Initial Operations

Subprograms falling into this category are only executed at the beginning of the terminal session. The flag 'initial_entry' controls their execution.

-- PROCEDURE check_required_files;

This procedure checks to insure that all main (disk drive A:) files necessary to sustain program operations are present. If a file is missing the program notifies the operator as to the filename of the required file and terminates execution.

-- PROCEDURE sign_on;

This procedure displays the RCS UNITREP SUBSYSTEM version number and the serial and transmittal status of the report currently in the workfile (Figure A-1).

b. Action Evaluation Subprograms

As discussed in Chapter III, eight program actions are permitted. Figures A-2 and A-3 detail the screen displays for action selection and the required additional prompts respectively.

-- PROCEDURE choose_action(action_is: OUT action_state);

This procedure is executed whenever control is passed to PACKAGE unitrep. All actions are mutually exclusive. The appropriate component of 'action_is' is set to TRUE.

```
-- PROCEDURE process_action(action_is: IN OUT action_state;
                            success: OUT BOOLEAN);
```

This procedure insures the legality and proper sequencing of requested actions and performs the preliminary operations such as format setting and data filename selection required to support that action. It consists of a multiple IF-ELSIF...ELSIF-ELSE-END IF construct which

evaluates the selected action. The output of this subprogram is used to drive a similar construct in the operative modules for processing the action on the area data. The constraints for each action are:

-- action_is(draft): The drafting document may be requested at any time during the report generation cycle. Draft information is normally based on the most recent report transmitted. However, if a report is in work in the system the operator may request draft information based on the pending report. This option is used to provide the releaser the detailed information required to evaluate the report with minimal reference to the instruction. Depending on operator selection the appropriate data file is determined.

-- action_is(neww): This action may be selected only if the workfile has been transmitted. If it has not been transmitted the operator is notified that only the modification, erase or log actions are allowed.

-- action_is(modify): Modification of data is only allowed for reports which have not yet been logged as transmitted. If the report in the workfile has been transmitted the user is notified to select the new action.

-- action_is(print): The print action may be requested at any time during the report generation cycle. The operator may request a print of either the pending or a past message. Past messages may be recalled by either serial or date-time group. Date-time groups are cross referenced to the appropriate serial. If a past message, the external disk data files are checked to insure they are available. If they are not the operator is notified. Prior to printing a valid message the user is given an option to select the print format.

-- action_is(log): Logging a message as transmitted is only allowed when the report in the workfile has not already been logged. The user is notified if there is an attempt to do this. When a valid action the user is prompted for the transmittal date-time group. This date-time group is entered with the report serial number in an external cross reference file (CROSSREF). The new data file name, B:UNITxxx is set based on the report serial for use in filing the report data on the disk.

-- action_is(erase): The erase action may only be selected when the report in the workfile has not been transmitted. The program erases the workfile by copying the data file of the last transmitted report. Therefore the external data files are checked to insure that those required are present. If they are not, the program terminates, notifying the operator which data file is required on the B: disk drive.

-- action_is(verify): Feedback verification may be requested at any time. The operator is prompted for the verification serial and the program searches the external data files for that B:UNITxxx required. The operator is notified if the data file is not present.

-- action_is(quit): This action terminates the program normally.

c. Area Selection

For draft or modify actions the program requires the selection of the appropriate UNITREP situation to direct the flow to the appropriate modules(s) (Figure A-4). Areas are automatically limited to those implemented (LOCAL, ADMINISTRATIVE, AIR) with a reselection capability provided. PROCEDURE choose_area(area_is: IN OUT area_state) provides the required constructs.

2. Module Control (unitrep(x))

The main body of PACKAGE unitrep and PACKAGEs unitrep1, unitrep2, unitrep3 provide the program control between these four modules. Program flow is based on the relationships between the selected action, the selected area(s) and the current execution point of the program. All information is contained in a program data file (STATUS) whose structure was discussed in relation to PACKAGE

urglobal. The basic operations, other than control, supported by each package are:

- -- unitrep: provides action and area selection and data file definition.
- -- unitrep1: provides access to all actions on the LOCAL area data.
- -- unitrep2: provides access to all actions on ADMINISTRATIVE area data and access to all actions on AIR data with the exception of modify.
- -- unitrep3: provides access to the modification of AIR area data.

The flow charts of Figures 4-6 thru 4-10 detail control of program direction.

3. File Handling (filerA/B)

The filer packages provide for checks of valid files, controlled program termination, workfile loading and storage and program status file loading and storage. The major difference in the files is that filerA manipulates files for the 'A' data structure while filerB manipulates files for the 'B' data structure.

-- FUNCTION valid_file(filename: STRING) RETURN BOOLEAN;

This function is used to check external disk files for existence to avoid abnormal program termination if an external file is not available. If a portion of the 'filename' is "UNIT", an 'A' or 'B' suffix will be appended (to distinguish between A and B data files).

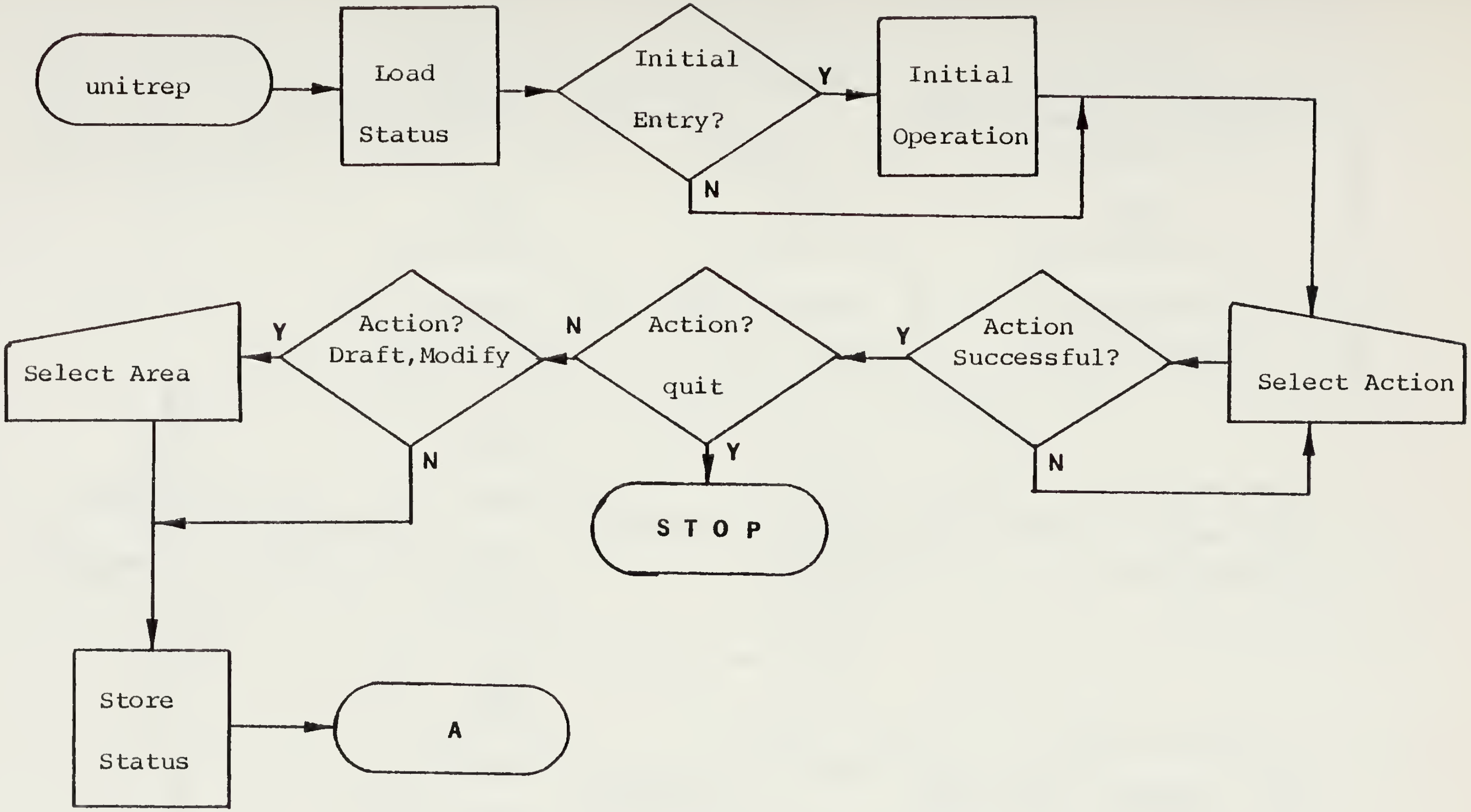


Figure 4-6. PACKAGE unitrep Action/Area Selection Flow Chart

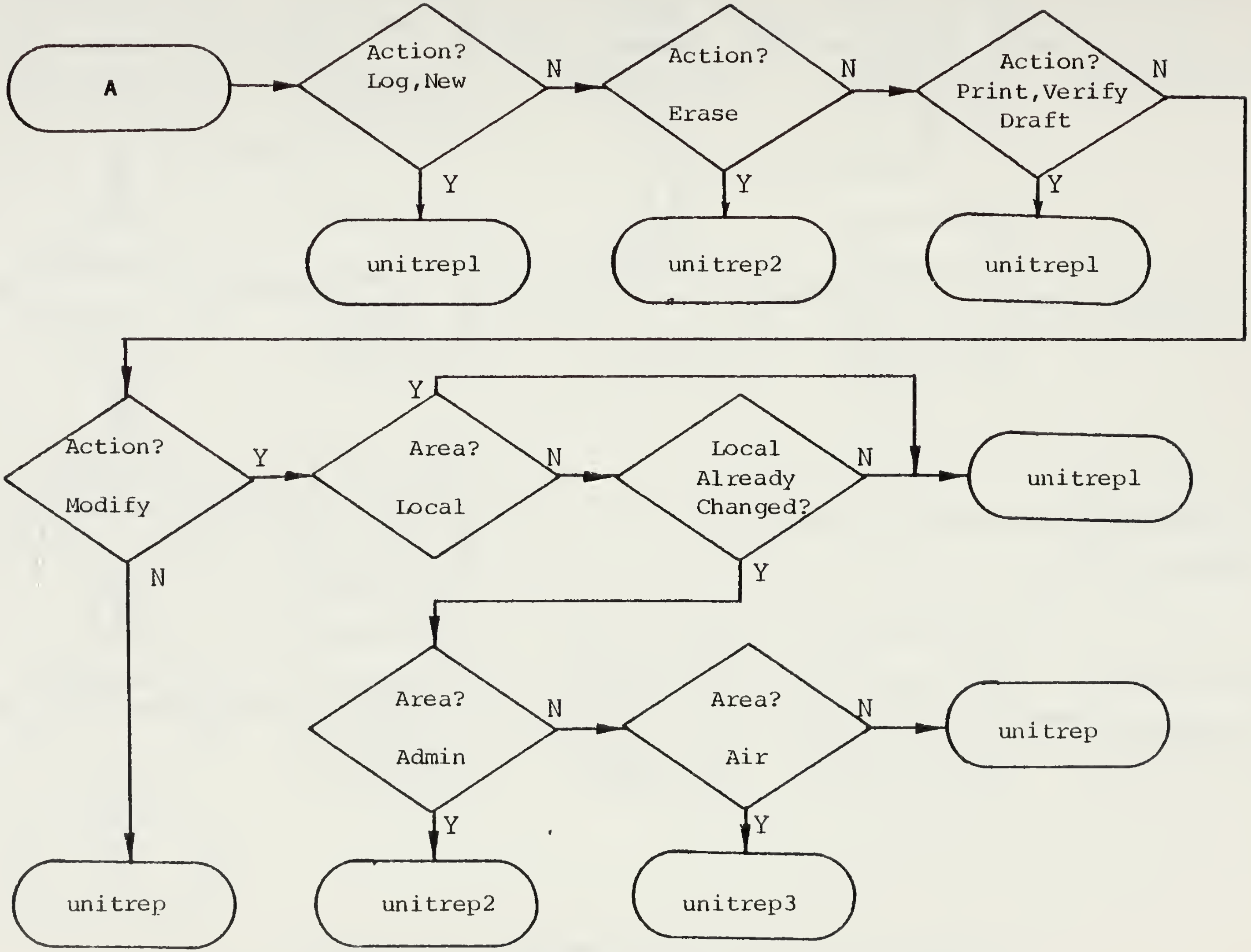


Figure 4-7. PACKAGE unitrep Control Flow Chart

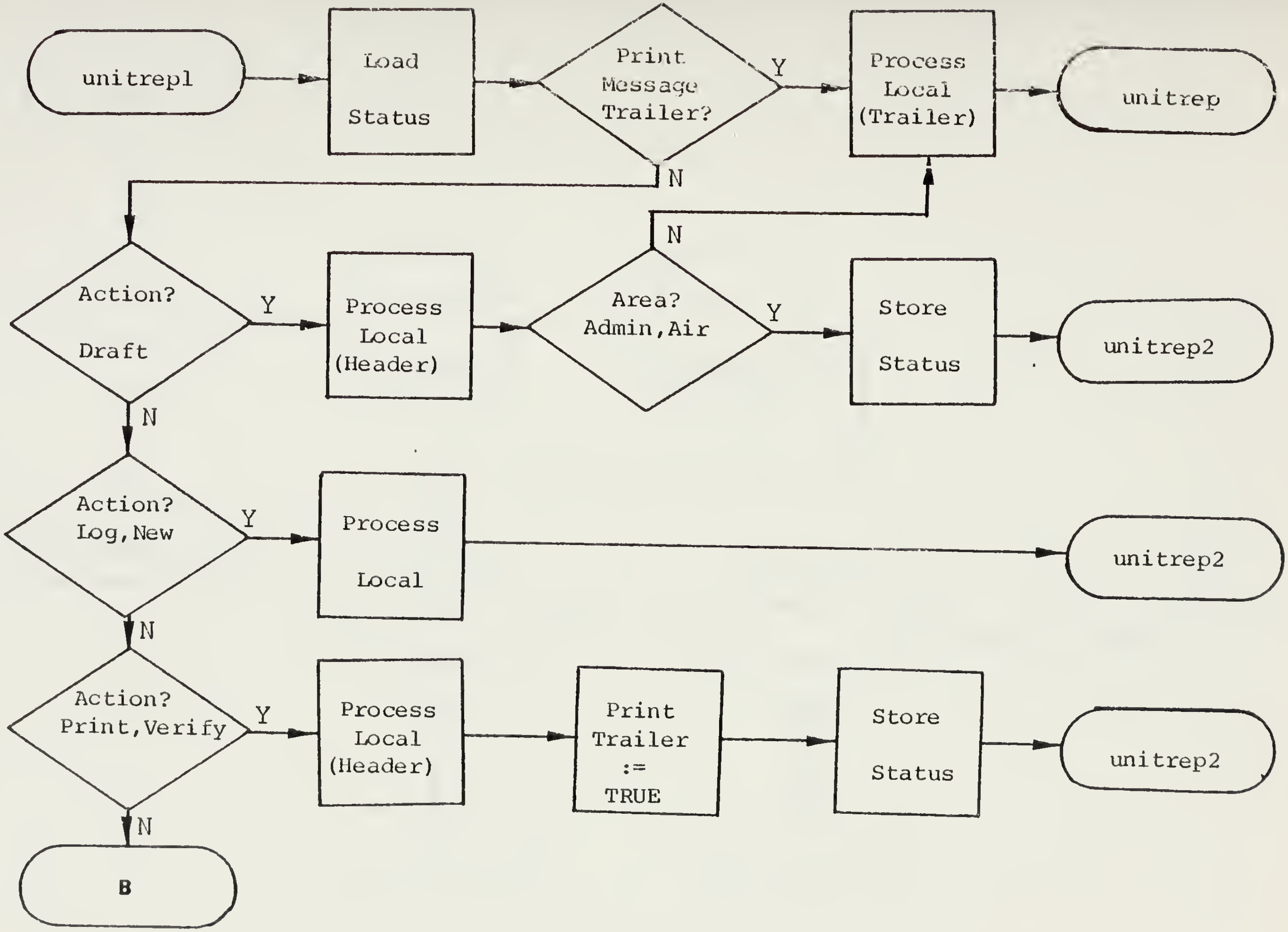


Figure 4-8. PACKAGE unitrepl Control Flow Chart

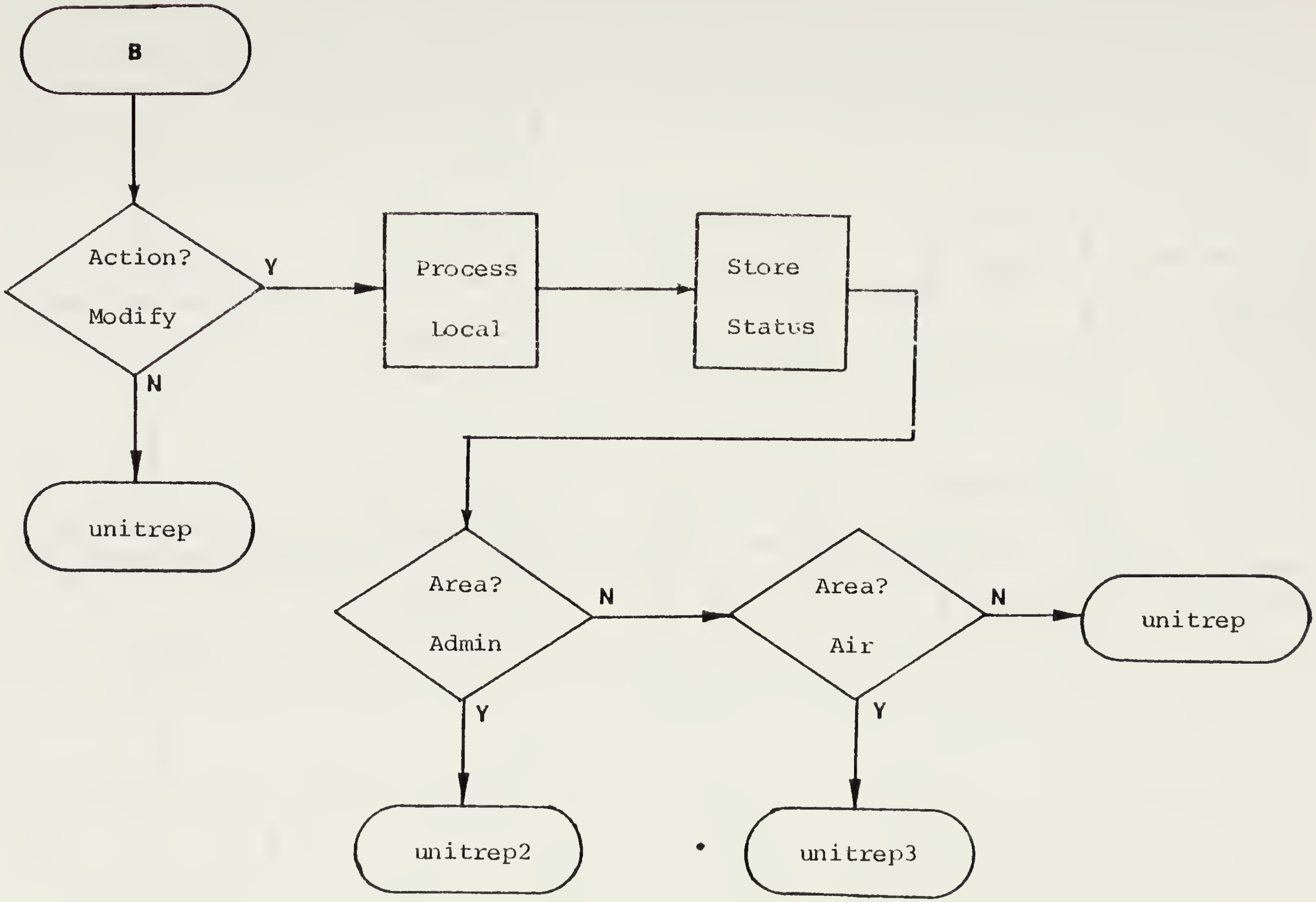


Figure 4-8. PACKAGE unitrepl Control Flow Chart (cont'd)

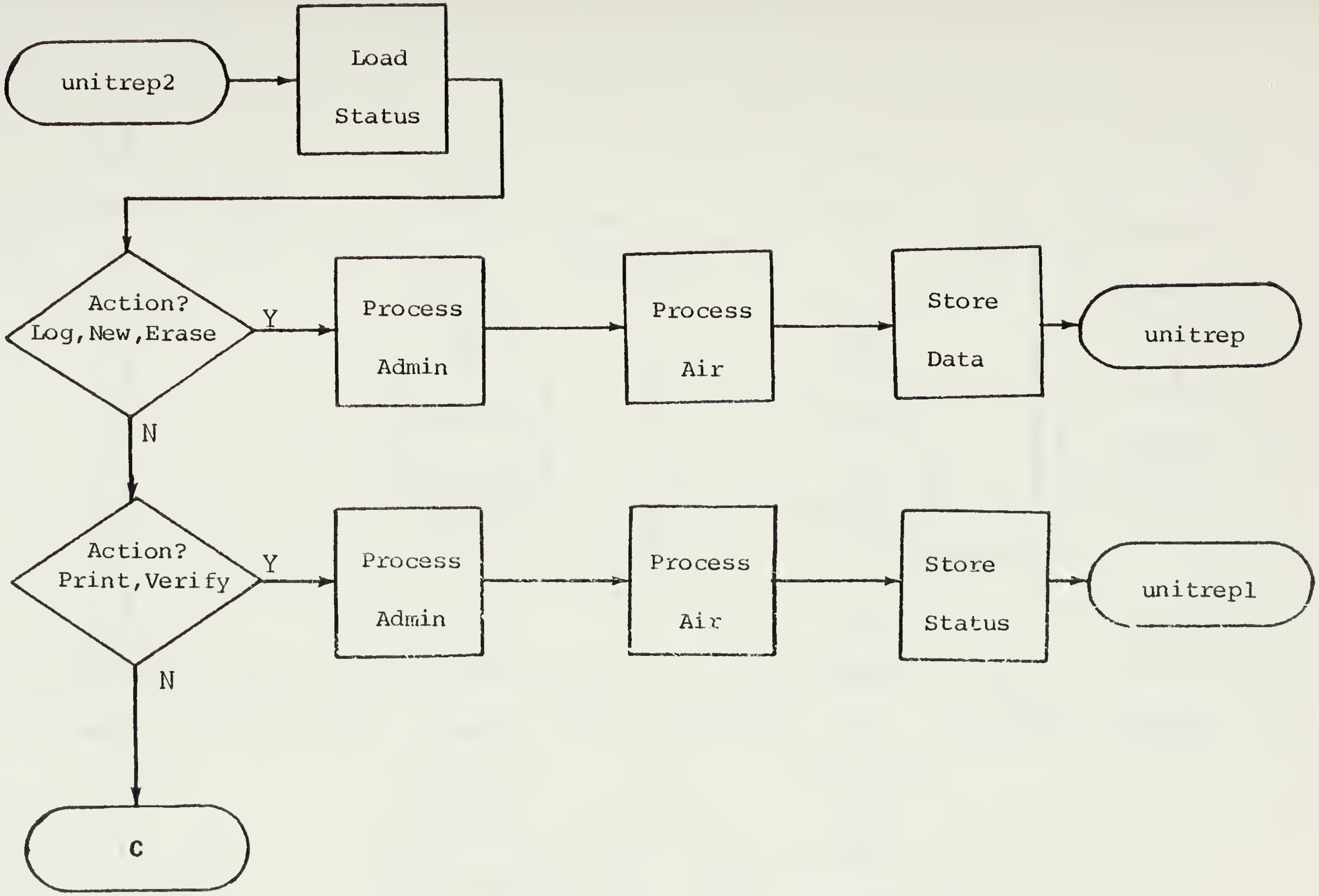


Figure 4-9. PACKAGE unitrep2 Control Flow Chart

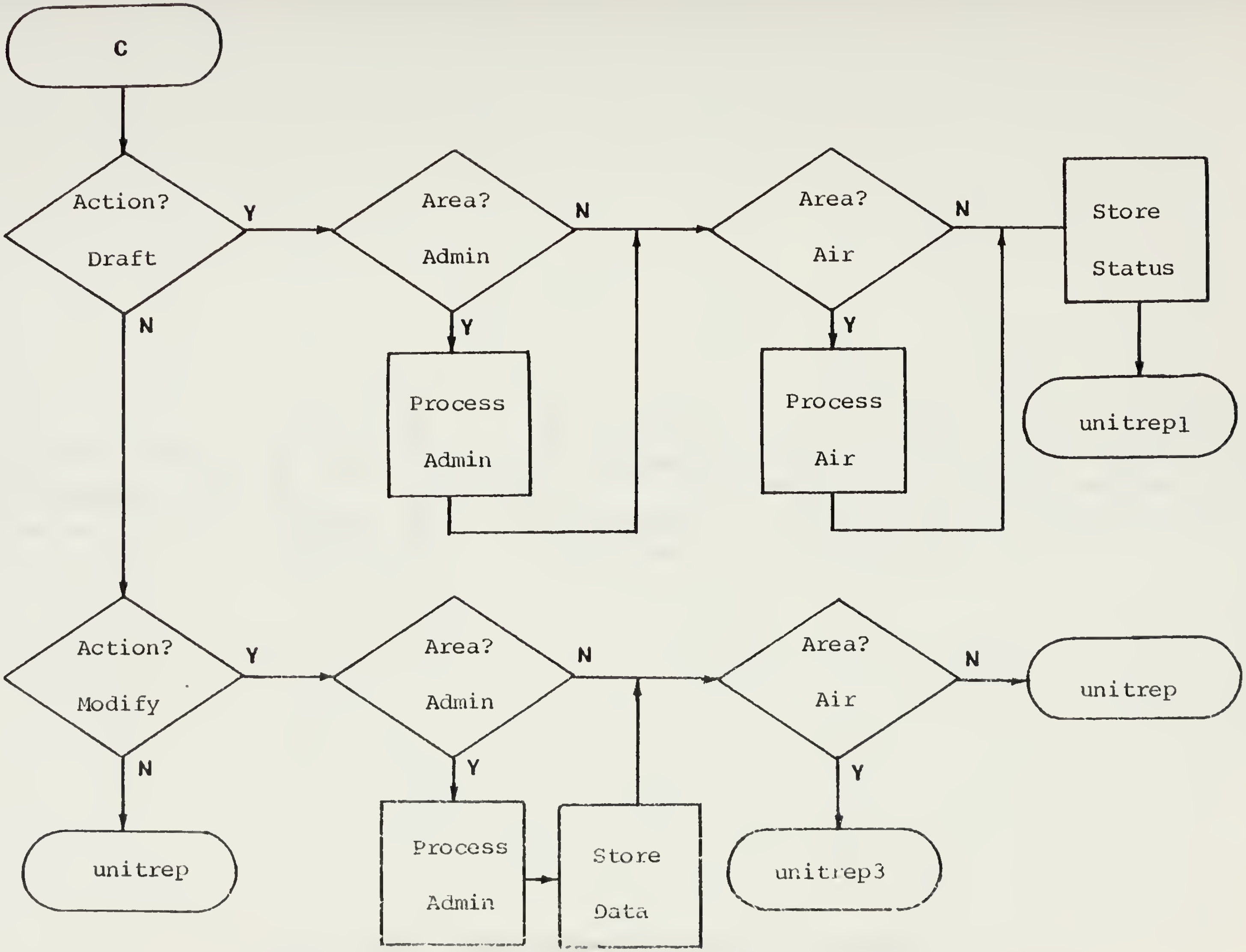


Figure 4-9. PACKAGE unitrep2 Control Flow Chart (cont'd)

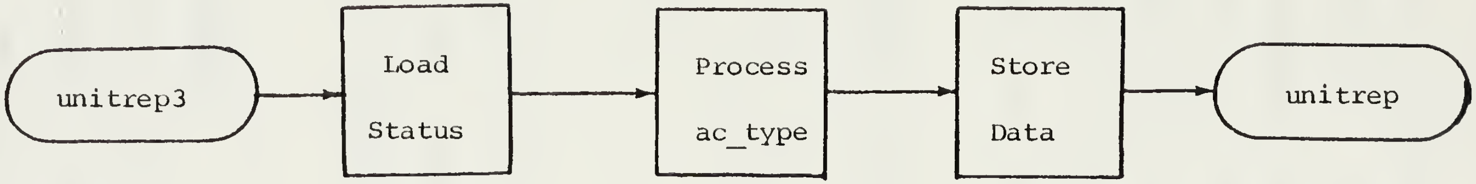


Figure 4-10. PACKAGE unitrep3 Control Flow Chart

```
-- TYPE reason_code IS (normal,file_error)
-- PROCEDURE terminate(reason: IN reason_code;
                                filename: IN STRING);
```

The terminate procedure is used for two reasons: a file error or a normal program halt. File error terminations are used only when external files are mandatory to continue program operation and are not available. The procedure provides a prompt which identifies the required file so it may be loaded or copied to the proper disk. Normal termination occurs only when the quit action is selected.

```
-- PROCEDURE load_file(filename: IN STRING);
-- PROCEDURE store_file(filename: IN STRING);
```

These procedures load/store the external UNITREP data structure (UNITxxxA/B) from/to secondary storage (disk drives) to/from main memory.

```
-- PROCEDURE load_status;
-- PROCEDURE store_status;
```

These procedures load/store the current program state from/to the STATUS disk file. Appropriate statements are executed to initialize all required variables for proper program operation.

F. PROGRAM OPERATIVE COMPONENTS

The operative components of the program are organized according to the basic UNITREP situations as defined in Chapter III. They are responsible for excecuting the operator selected action for the particular area data. To

maintain uniformity, common constructs were applied in the processing of these actions. However, each area also has unique requirements. These specific points of interest will be discussed.

1. Common Elements

There are basic concepts for which the implementation is similar, if not identical, for all packages. The general interface to an area package is PROCEDURE process_(area name). Within this subprogram only those statements required to support the operator selected action are executed. An IF-ELSIF...ELSIF-ELSE-END IF construct, identical in structure to that of the program control components, is used. A summary of the these operations follows.

-- action_is(neww): Resets all flags used to indicate a data set or field has been modified, deletes from the data structure all items tagged for deletion and resets the content of any data items which are dependent on the report occurence (e.g. message serial, precedence, date-time group).

-- action_is(log): Deletes any free text (.AMP) disk files created during the generation of the report.

-- action_is(print), action_is(draft), action_is(verify):

For each data set in the area a check is made, following the decision tree of Figure 4-5, to determine if the data set is to be printed. The decision tree is implemented by

three functions: PACKAGE urglobal, FUNCTION required_print is used for sets which can not be deleted; PACKAGE urair, FUNCTIONs print_with_delete and print_wo_delete are used for sets which may be deleted. If the set is to be printed the proper format is provided. If amplification comments for the set exist they are printed immediately following the set (PACKAGE urglobal, PROCEDURE print_comment). When requested, draft information is provided (PACKAGE urglobal, PROCEDURE draft_aid).

Operations for actions 'erase' and 'quit' are executed in the control packages. Modification actions are customized to the particular area and are discussed next.

2. General Modification Constructs

Modification is the only action which requires considerable operator inputs. As the desired output of the system is an error free message, many of the statements are concerned with insuring that data entered is valid. Each area modification section is divided into one or more subprograms which process the information for a specific area display. The displays are directly aligned to the draft document the system generates. The common elements for a display modification subprogram are:

-- PROCEDURE (area/subarea)_mask;

This procedure generates the particular area/subarea screen display. This display consists of all data sets or

information items for the area/subarea. Common elements for the screen display are:

-- Data Set Access Point ['( )']: Each data set or information item has an access point to which the cursor positions automatically. Data set modification proceeds in sequence from the top of the display to the bottom. The operator enters one of a defined set of access codes (keyboard characters).

-- Data Set Name: This is the plain language expansion for the data set identifier (e.g. 'Physician Status' for the MEDIC data set).

-- Data Set Fields: Each field of the data set is annotated with its plain language meaning. The field may consist of a series of dashes ('---...---') of the assigned maximum field length, a template (e.g. 'ddhhmmZMMMyy for date-time groups) or ,for limited selections, a series of alternate selection points ['( )' or '-'] labeled with the corresponding data field values. -- On Screen Prompts: An abbreviated table of the valid access codes allowed for the display.

-- PROCEDURE fill_(area/subarea)[_mask];

This procedure fills in the corresponding area/subarea display. All data fields are annotated with the current content of the data structure for that data set. Alternate selection points are marked as '(X)'. Items changed for the pending report are marked with a '*' next

to the access point. Items to be reported as deleted are marked with a 'D' next to the access point. If a free text set (AMPN) has been created for the data set an 'A' is marked next to the '*' ('AMPN' sets apply only to data sets which have been changed).

Once the fill subprogram has been executed the system is ready to process the operator modifications to the current data structure. The cursor automatically positions to the access point of the first data set of the display. Each data set is processed through a LOOP and enclosed CASE construct. The loop is labeled 'modify_(data set identifier):'. The program waits for the operator to enter the access code and then executes the statements required:

-- space or carriage return: Access to this data set is not desired. Position the cursor to the next data set.

-- 'X' or tab: This code allows further access to the data fields or information items within the data set. If the access point is for a data set the cursor is positioned to modify the required data fields. Data field entries are checked to insure they properly correspond to the data field items (e.g. numerics only, date-time groups, free text, valid tabularized codes). Improper input generates an error message. Errors are required to be corrected before proceeding with the modification. If the access point is for an information item with other

subdivisions, further access is granted to select the components of the item which are to be modified.

-- 'C': This code allows access to a data field for change purposes. It is used for data sets which are components of an information item accessed by an 'X' which may also be deleted. A check is first made to insure the set has not been marked for deletion. If data change is allowed, similar operations to insure correctness of data input are performed.

-- 'D': This code marks a set to be reported as deleted. A check is first made to insure the set has not been changed as deletions apply only to previously transmitted data sets.

-- 'R': This code allows the operator to reset a deletion flag if the set has been marked for deletion incorrectly.

-- 'A': This code allows the operator to create a free text (AMPN) set to provide amplifying comments to the data set. As amplification applies only to a data set which has been changed a check is made to insure this.

-- '?': This code will permit accessing of HELP information, if it exists, for the data set through PACKAGE urglobal, PROCEDURE help. If a corresponding look-up table is available for a data field item that option is also provided through PACKAGE urglobal, PROCEDURE view_table .

-- other inputs: All other operator inputs are rejected with an error message to reselect.

The enclosed loop structure for the case statement allows for multiple operator actions on the same data set without the necessity of repositioning the cursor manually (e.g. a HELP access '?', followed by a change access 'X' or 'C' and modification of information, followed by an amplification access 'A'). The cursor is positioned to the next data set access point only when the user has indicated that access to the current data set is no longer desired (by space or carriage return).

When all data sets for the display have been sequentially processed the operator is given the option to reselect the current display (e.g. if an error has been made) or to continue on to the next display.

3. Local Information (urlocal, urlocal1)

Local information is grouped in three displays: Infrequently Changed Items (Figure A-6), Message Information (Figure A-7) and Miscellaneous Items (Figure A-8). Within the Miscellaneous Information the Operation (OPER) and Exercise (EXER) sets are mutually exclusive. This exclusion is enforced by an IF-ELSIF-ELSE-END IF construct to automatically position the cursor to only valid access points depending on the current data. Position (POSIT) is an example of a set which is partially dependent on unit type, required by ships and submarines but only when changed for other units. The IN operation on the enumeration type

'unit_type' is used to control automatic selection. An additional check is made to determine if the unit is 'EXEMPT' from reporting position. The Declassification (DCLAS) set is dependent on message classification. Again the access point is controlled by an IF IN construct.

4. Administrative Information (uradmin)

The ADMINISTATIVE Information consists of a single display (Figure A-10). The Administrative sets are not subject to deletion. Within the display only the Physician Status (MEDIC) set is dependent (on unit_type). Access to selection is controlled by by an IF-ELSE-END IF construct. The unit activity code (ACTIV) is cross checked [Ref. 10: Table 3-6].

5. Air Information (urair, urair1, urair2, urair3)

As opposed to the LOCAL and ADMINISTRATIVE Situations, the AIRCRAFT and CREWS STATUS data sets may be submitted a number of different times depending on the aircraft type and location. Additionally, they may be deleted. The variable number of sets was discussed earlier in this chapter in relation to the data structure. All actions other than modify are processed in PACKAGE urair. The modification of air information is accomplished by the other three packages. Three displays are used:

-- Authorizations (Figure A-11) details the aircraft type, aircraft and crews authorized and the number of different locations at which the aircraft are deployed.

-- Location (Figure A-12) lists the locations at which the type aircraft are deployed.

-- Status (Figure A-13) lists the aircraft, crews and reconnaissance capabilities of the particular type aircraft at the selected location.

Access to a particular type of aircraft listed on the Authorization page is granted by an 'X'. A change to the aircraft type is permitted by the 'C' access code (used for error correction only). The aircraft type is cross checked [Ref. 10: Table B-2]. Modifications for that type aircraft are allowed to the Aircraft and Crews Authorizations by a 'C', 'D'-'R', and 'A'. Display of the Locations is granted by an 'X'. On the Location page a pseudonym, HOMEPLATE, is used to denote the location as reported in the Position (POSIT) set. Within the Status page all status information for the location may be deleted at the location access point or items may be deleted, changed or amplified individually. Component fields of both the aircraft status (AIRSTAT) and crews status (CREWSTAT) are sum/total checked (Possessed := FMC+PMC+NMC, Formed >= Ready). Reconnaissance Capabilities are cross checked [Ref. 10: Table B-3]. As the entire reconnaissance capability (RECON) set must be reported whenever changed, individual data field modification (addition or removal) is permitted.

A degree of flexibility is built into the system by using a display format which is based on the maximum

aircraft types/locations declared in the data structure definition. This will allow a maximum of seven aircraft types and seven different locations for each type without the necessity of changing the screen displays. It should be noted that current memory size (64K) limits these values to a maximum of four and four respectively. However a different combination of types/locations would be permissible if the present limitation (approximately 3K) for air data was not exceeded (e.g. 2 types/7 locations, 6 types/2 locations, etc.).

G. MISCELLANEOUS

PACKAGEs initialA and intialB are used to set the workfile (UNIT000A, UNIT000B), the program status file (STATUS) and the serial/date-time group cross reference file (CROSSREF) for initial program operation. They would not be required at the unit level as these files would be distributed on the program master disk. The status file is set for normal system operations. The workfile is set as:

Message Precedence - Priority
Message Classification - Confidential
UNITREP serial - 001
ACTIV - OP (General Operations)
MEDIC - ONBCARD
REPORG - CNO
VERIFY - VALID

All numeric items are set to zero, all character fields are blank. On receipt the operator corrects these values as required.

V. CONCLUSIONS

Limited testing of the ROS UNITREP SUBSYSTEM was conducted at the Naval Postgraduate School. Test participants were of the mid-level officer grades (O-3/O-4) normally responsible for the drafting of the UNITREP. The test consisted of simple data entry from the drafting form and real-time report generation during which the test participants used the built-in help facilities without referencing the draft document. Seven simulated reports were generated by each participant during two terminal sessions. System familiarization •required approximately forty five minutes. Refamiliarization time was minimal. The participants were required to fill in the system-provided draft form for the first report after which the drafting was optional. As experience with the system was gained the participants tended more towards real-time report generation, forgoing use of the draft document. Simple data entry after the drafting of the report required approximately fifteen minutes. Real-time report generation required approximately thirty five minutes. No premature terminations were observed during the tests.

Due to program modularization, disk access time (during which the operator must wait) is sizable. Each module requires approximately one minute to load into main memory.

Thus, four minutes during each modification cycle are operator 'dead time'. Printing a message requires two minutes; the draft document requires five minutes.

Several participant suggestions were incorporated during the test period (e.g. the addition of the '*' to the screen display to note an item change). Modifications were easily accomplished largely due to the uniformity used in code expressions and the localization of the procedures. No attempt was made to optimize the current UNITREP implementation. There remain items which, if revised, would provide more efficient operation or clearer uniformity (e.g. the use of a common table comparison procedure). Using the techniques developed in Chapter III it should be possible to implement the remainder of the UNITREP within a short period of time and apply a similar methodology to the other NRS reports. However, the limitations of JANUS/ADA (version 1.4.5) would restrict the program to multiple disk operations.

Many additional areas for increasing the system's flexibility and/or utility remain to be explored:

-- The creation of a module dedicated to the preparation of non-formatted messages. The module should integrate the heading and trailing items of a Joint Messageform (DD-173), provide for a page printer, page alignment procedures and incorporate a full screen text editor.

Portions of the module should be accessible to the ROS for preparation of the AMPN and RMKS free text data sets and for the printing of the message.

-- Examination of alternate methods of storing the program generated data. The use of a data structure based upon static record types is inflexible. A better method would seem to be writing the data to a disk file in the message format as it appears on the printed message copy.

-- Examination of the feasibility of transfering the report information to the message communication center via a network system or via disk transfer. Either medium is capable of compatibility with communications equipment. A special purpose module could be developed for the message center to incorporate items required for broadcast (e.g. prosigns, serialization, circuit instructions, conversion to transmission code, special formatting requirements). The necessity of punching a paper tape or retyping the information on a video display terminal (as is the current practice) would be eliminated as would the possibility of introducing errors.

-- Creation of a master design program to assist in the development of programs of this type. Since many of the steps involved in customizing a program for a particular report are redundant, it should be feasible to develop a master program to assist in generating the necessary code. This program would require such basic information

as the data set identifier, the component fields, the type and specifications of data entered in each field (e.g. numeric, alphanumeric, limited selections, table look-up, field length) and the more complex dependencies between data sets. One of the most time consuming aspects of developing the UNITREP SUBSYSTEM was adjusting screen formats for the access points, data sets names and the data set components, and maintaining proper cursor positioning when executing fill and modification subprograms for the particular display. An interactive facility could be developed to integrate the formulation of a screen display with the required system operations on the data set.

The UNITREP itself proved to be a pivotal message. It's content, especially in the area of Unit Combat Readiness Assessment, is highly dependent on information available and maintained by other sources (e.g. Personnel - squadron roster, OPNAVNOTE 1000/2; Major Equipment - CASREPs; Aircraft Status - VIDS/MAF system). While it is simple enough to create a program to check format and valid data type entry, to develop a system to sift the raw data necessary to calculate a unit's readiness is much more complex. If a desire of the CCRS program is to remove subjective human appraisal from the assesent of unit readiness it will be necessary to integrate these other

sources at the unit level and determine the raw data to be used in this calculation. Such a system would allow event driven automatic reporting but would be extremely sophisticated (and costly).

Regardless of the level of detail desired, if the ROS is to have widespread distribution, hardware and software commonality will be required to reduce life cycle costs. It would not be cost effective to maintain a software library of similar programs for each make and model of microcomputer in the fleet. In addition to the software support aspects, the cost of the hardware and its maintenance could be reduced through the defense acquisition process.

Commercial microcomputers may not have long lifetimes in comparison to their military counterparts, but their economy and universal utility more than counterbalances this. The shorter service life may in fact be a benefit as commercial equipment will always be at or near the state of the art and at a competitive price. Every effort should be made to procure, for each unit, a general purpose microcomputer system and the software tools to effectively make use of it.

APPENDIX A

ROS UNITREP OPERATOR MANUAL

The ROS UNITREP SUBSYSTEM provides the message orignator a tool for the drafting, preparation and review of UNITREP messages. The ROS UNITREP Operator Manual is subdivided as:

I. BACKGROUND

II. GENERAL INSTALLATION PROCEDURES

III. GENERAL PROGRAM OPERATION

IV. PROGRAM USAGE

V. SUPPORTED UNITREP REPORTING SITUATIONS

- A. LOCAL INFORMATION
- B. ADMINISTRATIVE STATUS
- C. AIRCRAFT AND CREWS STATUS

I. BACKGROUND

The National Command Authority and the Joint Chiefs of Staff require large quantities of detailed information in order to maintain the correct status of forces throughout the world. Units are required to submit certain reports (e.g. CASREP, UNITREP and MOVEREP), normally via message, to keep their superiors informed of the organization location and status. This information is processed and made available

to necessary commands through the World Wide Military Command and Control System (WWMCCS). The Navy intends to consolidate operational reports under the proposed OPNAVINST 3503.x series: Navy Reporting Structure (NRS) Operational Reports. The instruction establishes requirements for reports, data set formatting and definitions that are compatible with other US/ALLIED agencies.

Message reports are normally processed by a computer at one of four WWMCCS sites. Such automated handling is extremely sensitive to strict formatting requirements in order to correctly interpet received message data. Recent studies have shown an increasing trend in format and data error rates as a result of stricter formatting requirements in current reports. This results in delays as the information is queued to an operator for manual error correction. Greater error rates may be anticipated when the proposed instruction is released unless a means of assisting the originator in their preparation is employed.

The concept behind the Report Origination System (ROS) is to provide the message originator a microcomputer based aid for the preparation of highly formatted reports. Ultimately the ROS SYSTEM will integrate modules for all the NRS reports: CASREP, STAFFREP, EMSREP, UNITREP, SURFMOVE and SUBMOVE. The ROS UNITREP SUBSYSTEM, version 1.1, is a partial implementation of this concept, using the Ada

programming language, directed at supporting UNITREP report submission of Administrative Status and Aircraft and Crews Status Reports (CPNAVINST 3503.5 sections 4 and 6). The SUBSYSTEM is designed to assist three levels in the report generation cycle; the system operator, the message drafter and the message releaser.

To assist the system operator the program is 'user friendly', navigating through the report by a combination of screen prompts, menu selection and error messages. The data set formats are presented in an understandable manner with access limited to those that apply for the type message. All information is verified as it is entered, which eliminates data inputs obviously in error (e.g. alphabetic characters where numbers are required, incorrect date-time groups, invalid entry from a table). The drafting document which the system provides is closely aligned to the display screens. A message formatted output copy of the report is provided in addition to the ability to reproduce printed copies of past messages.

The drafting document provides the drafter with a quick means of determining what is currently reported, what is required to be reported and what the source for that information is. A 'fill in the blank' format is used for each data item. The SUBSYSTEM provides the actual formatting. When required to perform a feedback verification

of information held by the WWMCCS database, the SUBSYSTEM can automatically provide all data submitted as of a particular serial report. The drafter is provided on line 'HELP' information to aid in real-time report preparation if so desired. Limited information included in the governing instruction (e.g. look-up tables, reporting requirements) is made available on screen.

For the releaser there exists an option to provide a draft document which consists of all data as is currently reported or that will be reported with the message. The data is logically grouped with reporting requirements noted and an explanation of any coded items (e.g. unit activity code, reconnaissance capability).

II. GENERAL INSTALLATION PROCEDURES

The RCS UNITREP SUBSYSTEM, Version 1.1, is designed to run on a Z80/8080 based 8-bit microcomputer system with 64K of main memory and dual single-sided single-density floppy disk drives running under Digital Research's CP/M system (Version 2.21). The current program is customized for an Altos Computer System ACS 8000-1 with a DATAMEDIA ELITE 2500 keyboard and terminal. A basic understanding of microcomputer system operation is advisable.

The following steps describe the procedures to be followed upon INITIAL RECEIPT of the program:

1) Copy the UNITREP master disk to a reformatted disk

and label it 'UNITREP MASTER'. Store the master disk in a safe location following normal safety procedures in the handling of floppy disks.

2) As the ROS UNITREP program must run on a CP/M based system, it is necessary to copy your version of CP/M to the first two tracks of the reformed UNITREP MASTER disk by issuing the SYSGEN command. It is also necessary to copy the basic Input/Output System for peripheral control, CBIOS64.COM, to 'UNITREP MASTER' from any existing disk.

3) Label another reformatted disk 'UNITREP FILES xxx-yyy'. Place the serial number of the UNITREP you intend to generate first in the xxx slot. Each disk will hold approximately 25 UNITREP messages.

4) Insert the UNITREP MASTER disk in the systems A: drive. Insert the UNITREP FILES disk in the B: drive.

5) Observe that the required files are on the UNITREP MASTER disk by typing the directory command, DIR. The required files are:

```
UNITREP.COM   UNITREP1.COM   UNITREP2.COM   UNITREP3.COM
UNIT000A      UNIT000B       STATUS
ACTION        INFO           ERRMSG         CROSSREF
TABLE.B-2     TABLE.B-3      TABLE.B-6
DRAFT.TXT     HELP.TXT
```

The operator is not required to know the function of each file but should be aware that these files are necessary to the proper operation of the program.

6) Copy the two files UNIT000A and UNIT000B to the UNITREP FILES disk on the B: drive. The purpose of these files will be explained later. (This should always be done when switching in a new UNITREP FILES disk)

III. GENERAL PROGRAM OPERATION

Since the final output of the ROS UNITREP SUBSYSTEM is a formatted, hard copy message, only characters that are compatible with both Optical Character Reader (OCR) and BAUDOT (a message transmission code) are accepted from the keyboard. The operator is automatically restricted to these type of characters. If the operator attempts to put in any unacceptable characters an aural tone will sound. Lowercase letters are automatically converted to uppercase.

Program operation is controlled through a combination of menus and screen 'access points'. Menus require the operator to select one of the displayed options. Access points are screen locations to which the cursor is automatically positioned (noted by the symbol '( )'). Access points act as guards to a particular data set or information item. A set of valid access codes (e.g. 'C' to change, '?' for help) corresponds to each access point and is displayed at the bottom of the display. The access codes determine the operations required. If an invalid key is selected the operator is notified by an aural tone and when appropriate, an error message denoting the correct key to depress.

Throughout the program whenever an <X> is indicated the 'X' or the TAB key may be depressed. Whenever an <ENTER> is indicated the CARRIAGE RETURN or SPACE BAR may be depressed.

A premature ABORT of the program is available by depressing the CONTROL and 'C' keys simultaneously (CTRL-C).

*** NOTE ***

A user commanded abort will most likely result in the loss of all data generated since the last entry to the main (ACTION) menu.

****

For many data items a limited amount of on line HELP information is available. This is obtained by typing a '?' at the data set access point. If HELP is not available an aural tone will sound.

A brief discussion of the inner operations of the SUBSYSTEM is required to acquaint the operator with some terms frequently used in the program displays. The SUBSYSTEM's Current Workfile are the files UNIT000A and UNIT000B. These two files hold all the information on the unit's status as of the current moment. Once a message has been prepared AND logged as transmitted, the SUBSYSTEM creates files UNITxxxA and UNITxxxB on the B: disk (UNITREP FILES), where xxx is the serial number for that report. When the system is asked to retrieve old information it will search the B: disk for the required files. If they are not present the operator will be notified.

IV. PROGRAM USAGE

To execute the UNITREP program type 'UNITREP'.

The first screen to appear at the start of the terminal session is the SIGN ON display (Figure A-1) which gives the version of the program and the status of the current message in the workfile.

A. PROGRAM ACTIONS

Following the SIGN ON display is the ACTION MENU (Figure A-2). This display offers the operator the various program options. The prompt area provides additional information to the operator to support the requested action. Error messages are displayed in the message area. The ACTION MENU will always be redisplayed with the required action has been completed. A brief discussion of the purpose and prompts for each of the desired actions follows.

1. Prepare UNITREP Draft Document

This option allows the operator to prepare a draft document for assistance in the preparation of a UNITREP or for review of the realeasing authority. The draft document consists of all the information as currently reported by the unit, an explanation of any coded entries, the fields and limited explanations/sources for each data set. The draft document closely follows the screens presented in the ROS UNITREP SYSTEM. The draft document may be used as a working

```
                    R C S   U N I T R E P   S U B S Y S T E M
-------------------------------------------------------------------------------
SIGN ON - RCS UNITREP SUBSYSTEM - VERSION 1.1 - JAN 83

The Current Workfile is UNITREP Serial --- .


It HAS been logged as transmitted: DTG ddhhmmZMMMyy .
                  (or)
It HAS NOT been logged as transmitted. (Modification Allowed).






          <ENTER> to Continue, ENTER <?> for HELP Information:
-------------------------------------------------------------------------------
```

Figure A-1. RCS UNITREP SUBSYSTEM Sign On Display

```
                    R O S  U N I T R E P  S U B S Y S T E M
                                                            Serial xxx
-----------------------------------------------------------------------
ACTION MENU

      -1- Prepare UNITREP Draft Document
      -2- Initiate New UNITREP
      -3- Delete/Modify/Add Data to Pending UNITREP
      -4- Print Message Format Hard Copy of a UNITREP
      -5- Log Pending UNITREP as Transmitted
      -6- Erase Pending Unitrep
      -7- Prepare Verfication Summary
      -8- Quit

                    ENTER the desired action:
-----------------------------------------------------------------------
(prompt area)



-----------------------------------------------------------------------
(message area)
```

Figure A-2. ROS UNITREP Action Menu Display

copy by marking the appropriate areas for change and filling in the corresponding data fields.

The operator is provided a prompt (Figure A-3a) if the current workfile has not been logged as transmitted (action -5-). The first selection allows the operator to obtain the draft copy for the releaser after all data provided by the drafter has been entered. The second selection permits the creation of another drafting document for the current report. If the current workfile has already been transmitted the last reported status as reflected in the SUBSYSTEM will provide the draft information without prompting.

The next prompt reminds the operator that he must select the UNITREP SITUATION for which draft information is required. As discussed in OPNAVINST 3503.5 paragraph 1.5, a normal UNITREP message is concerned only with a small subset of the possible UNITREP data sets. Therefore data sets are grouped into subsets called SITUATIONs. In addition to the situations established by the instruction (e.g. Personnel Status, Major Equipment Status), Local Information and Type Commander Reports areas have also been identified to group data sets. The UNITREP SITUATION MENU (Figure A-4) is then displayed. The operator selects those situations for which drafting information is desired. LOCAL information (message addressees, classification, priority, etc.) will always be provided. All other SITUATIONS must be requested.

A sample draft document is shown in enclosure A-1.

```
------------------------------------------------------------------------
The Current Workfile has not been transmitted. Do you want ...
     ( ) Draft of the Current Workfile
     ( ) Draft of the last submitted UNITREP

                    ENTER <X> as required.
------------------------------------------------------------------------
```

a. Prompt for ACTION -1- : Draft

```
------------------------------------------------------------------------
Do you want a message format hard copy of ...
        ( ) Pending UNITREP
        ( ) Old UNITREP: serial ---  or  dtg: ddhhmmZMMMyy

                ENTER <X> and data as required.
------------------------------------------------------------------------
```

b. Prompt for ACTION -4- : Print

```
------------------------------------------------------------------------
To you want the message format in ...
        ( ) CCR format
        ( ) Standard Naval Message format

                    ENTER <X>, Ready Printer then <ENTER>
------------------------------------------------------------------------
```

c. Additional Prompt for ACTION -4- : Print

Figure A-3. RCS UNITREP User Prompts

```
------------------------------------------------------------------

              ENTER the transmittal dtg: ddhhmmZMMMyy

------------------------------------------------------------------
```

d. Prompt for ACTION -5- : Log

```
------------------------------------------------------------------
        Confirm you wish to erase UNITREP serial xxx
          (It has not been logged as transmitted)
                       ENTER (Y/N):

------------------------------------------------------------------
```

e. Prompt for ACTION -6- : Erase

```
------------------------------------------------------------------
ENTER the AS-OF verification serial ---
       (000 to restart)

------------------------------------------------------------------
```

f. Prompt for ACTION -7- : Verify

Figure A-3. ROS UNITREP User Prompts (cont'd)

```
                     R O S   U N I T R E P   S U B S Y S T E M
                                                            Serial xxx
-----------------------------------------------------------------------
UNITREP SITUATION MENU

            ( ) -0- Modify Local Information

            ( ) -1-  Personnel Status
            ( ) -2-  Administrative Status
            ( ) -3-  Unit Combat Readiness Assesment
            ( ) -4-  Aircraft and Crews Status
            ( ) -5-  Major Equipment Status
            ( ) -6-  Special Capabilities Status
            ( ) -7-  Increased Defense Readiness Status

            ( ) -8-  Reserve Augmentation Status
            ( ) -9-  Type Commander Reports


                     ENTER <X> in Desired Report Area(s)
-----------------------------------------------------------------------
(Message Area)
```

Figure A-4. ROS UNITREP Situation Menu Display

2. Initiate New UNITREP

Once a UNITREP has been logged as transmitted (action -5-) or erased (action -6-) this selection must be made to reset the current workfile for modification (recall that ALL UNITREP information for the unit is maintained in files UNIT000A and UNIT000B). Any attempt to modify (action -3-) a transmitted message raises an error which notifies of the proper sequence for continuing.

3. Delete/Modify/Add Data to the Pending UNITREP

This option allows access to the SITUATION MENU and then to the data set screens. The UNITREP SITUATION MENU (Figure A-4) is first displayed and the operator is prompted to select those areas which the pending message will affect. The Local Information area (-0-) will always be entered, regardless of whether or not it was selected, on the first modification of a new message. This is to set such items as the classification and declassification. All other SITUATIONS must be selected by the operator. The operator may continue to modify the message by repeated selection of action -3- until the message has been logged as transmitted (action -5-).

4. Print Message Format Hard Copy of a UNITREP

This action is selected when the operator requires the completed message as it is to be formatted for review/transmission, or for recall of past messages.

Two options are offered by prompt (Figure A-3b): Pending or Old. When an Old UNITREP is selected it will not provide amplification (AMPN) or remarks (RMKS) sets as these free text fields were purged from the SUBSYSTEM for memory storage reasons when the message was logged as transmitted (action -5-).

The operator is also given a print format option (Figure A-3c). OCR format gives 19 lines per page double spaced with a standard left margin and maximum line length of 69 characters. Addresses are indented to tab 25; FM, TO and INFO are printed for demonstration purposes. End of transmission, 'NNNN', is entered automatically at the end of the message. Standard Naval Message (Fleet Broadcast) format is 42 lines per the page single spaced and a maximum line length of 69. Addresses are indented to tab 8 preceeded by FM, TO and INFO as appropriate. End of transmission, 'BT', is entered where required.

5. Log Pending UNITREP as Transmitted

This action is to be selected once a message has been approved for transmittal AND RELEASED. Units assigning their own date-time groups should insure that a proof copy of the message has been obtained since logging a message as transmitted purges all of the amplification (AMPN) and remarks (RMKS) data sets from the SUBSYSTEM data structure. A prompt (Figure A-3d) is displayed and the operator enters the date-time group.

*** NOTE ***

Prompt entering of the transmittal date-time group is required as modification is allowed UNTIL THE USER has selected this action. This could cause the data for the SUBSYSTEM to be incorrect if subsequent modifications are made after the actual transmittal but prior to the logging as transmitted.

****

6. Erase Pending Unitrep

This action allows the operator to erase the pending UNITREP and return the SUBSYSTEM to the information state that existed as of the last transmitted UNITREP. The operator is prompted (Figure A-3e) to ensure an erase has been requested. Once a UNITREP has been logged as transmitted it may not be erased.

*** NOTE ***

The SUBSYSTEM erases a file by copying the last file logged as transmitted. It is recommended that when switching UNITREP FILE disks the LAST transmitted file, UNITxxxA and UNITxxxB, where xxx is the LAST transmitted serial number, be copied to the new UNITREP FILE disk. The old disk should then be appropriately annotated for the first file - last file sequence and stored as appropriate for its classification. If a file required to complete the erase is not present on the B: disk drive the program will terminate, notifying the operator that the required file must be copied to the B: disk drive.

****

7. Prepare Verfication Summary

When required by higher authority to perform a feedback verification (OPNAVINST 3503.5, paragraph 4.8) , this option allows the operator to enter an as-of verification serial (Figure A-3f). All data in the SUBSYSTEM

as of that serial number will be listed in message format to compare with the verification copy. Future versions will most likely change this to a date-time group under full NRS.

8. Quit

When the operator has completed the terminal session this action is used to terminate program execution. All modifications made will be preserved in the workfile. Therefore an entire generation cycle (draft, prepare, enter, review) need not be completed in a single setting. This frees the microcomputer system for other uses.

V. UNITREP SITUATIONS

As described in the proposed UNITREP instruction (OPNAVINST 3503.5 paragraph 1.5) certain changes in an organization's status require a UNITREP. The ROS UNITREP SUBSYSTEM is organized around logical groupings based on these changes. Whenever the operator selects to modify (action -3-) or draft (action -1-) the UNITREP SITUATION screen (Figure A-4) is displayed. The operator selects those situations that are required. Once the selection is complete the program executes those areas.

Each situation or area is divided into one or more screen displays. The area and subarea are noted near the top of the screen. Each individual information item or data set is displayed in plain english and numbered for

identification purposes. Modification to the current information is controlled by using access points, '( )', to which the cursor automatically positions. The operator enters an access code (keyboard character) and the appropriate action is taken or allowed. Valid access codes are displayed on each screen and are included for each item in Table A-1. A summary of the valid access code meanings:

'X' or TAB: grant access to the data fields for modification OR grant access to additional sub-access points which guard the actual data fields.

SPACE or CARRIAGE RETURN: no access required. go to next access/sub-access point.

'C': allow access to data field(s) for change purposes.

'D': mark this set for deletion.

'R': restore a set previously marked for deletion.

'A': amplification comments to be created (Figure A-5).

'?': HELP information requested.

any other character: incorrect input. operator notified by an error message and/or aural tone.

All data item modifications are brought to the operator's attention by the use of symbols next to the appropriate access point(s) of the display. When a data item has been changed it will be marked with an '*'. When a data item has been tagged for deletetion it will be marked with a 'D' (Use of the 'R' access code removes this 'D'). If amplification comments have been created an 'A' will be marked next to the '*'. This allows the operator to generate the report over a period of time without the difficulty

Table A-1. Data Item - Valid Access Code Correspondence

DISPLAY/ITEM	DATA SET	ACCESS CODES
LOCAL -INFREQUENT-		
-1- Unit Type		X
-2- Unit Identification	MSG-UNIT/ID	X
-3- Last Serial	MSG-UNIT/ID	X
-4- Feeder Report	MSG-UNIT/ID	X A ?
LOCAL -MESSAGE-		
-1- Addresses		X ?
Originator		X
Action		X (? within display)
Info		X (? within display)
-2- Classification		X
-3- Priority		X
-4- Declassification	DCLAS	X
LOCAL -MISCELLANEOUS-		
-1a- Operation	OPER	X A ?
-1b- Exercise	EXER	X A ?
-1c- Terminate	OPER/EXER	X A ?
-2- Position	POSIT	X A ?
-3- Remarks	RMKS	X (? within display)

Table A-1. Data Item - Valid Access Code Correspondence (cont'd)

DISPLAY/ITEM	DATA SET	ACCESS CODES
ADMINISTRATIVE		
-1- Command Change	COMMAND	X A ?
-2- Activity Change	ACTIV	X A ?
-3- Physician Status	MEDIC	X A ?
-4- New Reporting ORG	REPORG	X A ?
-5- Feedback Verification	VERIFY	X A ?
AIR -AUTHORIZATIONS-		
Access		X ? (C to change Type)
Aircraft Type	All AIR	
Aircraft Authorized	AIRAUTH	C A D R ?
Crews Authorized	CREWAUTH	C A D R ?
Deployed Locations		X ?
AIR -LOCATION-		
Access		X ? (C to change sp.)
Location	Many AIR	
AIR -STATUS-		
Location		D R ? (D/R all status)
-1- Aircraft	AIRSTAT	C A D R ?
-2- Crews	CREWSTAT	C A D R ?
-3- Reconnaissance	RECON	C A D R ?

```
                    R O S   U N I T R E P   S U B S Y S T E M

-------------------------------------------------------------------------
AMPLIFICATION SECTION - (data set name)
              or
REMARKS SECTION - PAGE x

       ----------------------------------------------------------
       ----------------------------------------------------------
       ----------------------------------------------------------
       ----------------------------------------------------------
       ----------------------------------------------------------
       ----------------------------------------------------------
       ----------------------------------------------------------
       ----------------------------------------------------------
       ----------------------------------------------------------
       ----------------------------------------------------------

<ENTER> to Continue, <C> to Change, <E> to Erase, <?> for HELP
                        or (When Change)
Position cursor to line to change and type over - ENTIRE LINE MUST BE CHANGED
-------------------------------------------------------------------------
(Message Area)
```

Figure A-5. ROS UNITREP Free Text Set Display

of remembering what has or has not been changed. All symbols are removed when a new report is initiated with the exception of items which are reported continuously (e.g. OPER or EXER when required).

*** NOTE ***

ROS UNITREP SUBSYSTEM, Version 1.1, implements only the LOCAL, ADMINISTRATIVE and AIRCRAFT AND CREWS STATUS situations. The operator is notified if an unimplemented area has been selected. Table A-2 provides a quick data set to screen display cross reference.

****

A. LOCAL INFORMATION

Local Information pertains to unit particular items. This section is subdivided into three (3) screen displays. Infrequently Changed Items, Message Information and Miscellaneous. All or part of the information in the Local area will appear for each message. A description of each display follows.

1. LOCAL -INFREQUENTLY CHANGED ITEMS- (Figure A-6)

These items are normally set when the ROS UNITREP SUBSYSTEM is initially received. After this only item -4- (Feeder Report) will require modification.

-1- UNIT TYPE: Self explanatory. Used to determine type unique reporting requirements (e.g. MEDIC for ships/submarines only).

-2- UNIT IDENTIFICATION: Used in the creation of MSGID and UNITID data sets (data field Message Originator).

Table A-2. Data Set - Screen Display Correspondence

DATA SET	SCREEN	ITEM
ACTIV	Administrative	2 - Activity Code
AIRAUTH	Air (auth)	Aircraft Authorized
AIRSTAT	Air (status)	1 - Aircraft
AMPN	ALL	Amplification
COMMAND	Administrative	1 - Command Change
CREWAUTH	Air (auth)	Crews Authorized
CREWSTAT	Air (status)	2 - Crews
DCLAS	Local (message)	4 - Declassification
DELETE	Air (auth) (status)	delete this item
EXER	Local (misc)	1b - Exercise
MEDIC	Administrative	3 - Physician Status
MSGID	Local (infrequent)	4 - Feeder
OPER	Local (misc)	1a - Operation
POSIT	Local (misc)	2 - Position
RECON	Air (status)	3 - Reconnaisance
REPORG	Administrative	4 - New Rep Org
RMKS	Local (misc)	3 - Remarks
UNITID	Local (infrequent)	4 - Feeder
VERIFY	Administrative	5 - Verification

```
                        R O S   U N I T R E P   S U B S Y S T E M
LOCAL
------------------------------------------------------------------------------
                          -INFREQUENTLY CHANGED ITEMS-

( ) -1- Unit Type:
        ( )Ship   ( )Submarine   ( )Air   ( )Shore   ( )Other


( ) -2- Unit Identification - OPNAVINST 3503.1 APP. C
        ------------------------------


( ) -3- Last Serial ---


( )     -4- Message to be Prepared as Feeder: [YES/NO]
                (UNITID vice MSGID data set)


ENTER <X>/tab to Select, <ENTER>/space for Next, <A> to Amplify, <?> for HELP
------------------------------------------------------------------------------
(Message Area)
```

Figure A-6. ROS UNITREP Local Infrequent Display

-3- LAST SERIAL: This item should only be changed when first receiving the ROS UNITREP SUBSYSTEM to synchronize the SUBSYSTEM with the unit's UNITREP serial number. A two line error message warns the operator not to change this value except for this reason. Three possible situations dictate the procedures to be followed:

a) Newly activated unit, first serial 001: No action is required as the SUBSYSTEM automatically begins serial numbers at 001.

b) Reactivated unit: Enter the serial number of the last UNITREP transmitted by the unit (it should have noted deactivation). Fill in UNITREP entries normally.

c) Currently active unit: Prior to generating the unit's next UNITREP it is necessary to initialize the SUBSYSTEM data structure. This is accomplished by simulating the unit's last report as a complete complement of data set submissions.
Enter the unit's last transmitted serial minus 1. (e.g. last serial 473 - enter 472). This resets the SUBSYSTEM current serial to that of the last transmitted serial. COMPLETELY fill in all UNITREP SITUATIONS as they existed as of the last serial. When all entries have been confirmed, log the message as transmitted (action -5-) with the last serial number's date-time group. After Initiating a New Report (action -2-) the SUBSYSTEM and unit serials will coincide.

-4- MESSAGE TO BE PREPARED AS A FEEDER REPORT: This item determines if a message should be prepared with a MSGID or UNITID data set and, as explained in OPNAVINST 3503.5 paragraph 2.5, depends on whether the unit is the releasing authority or if the message is an input to another unit which will release it in a combined report. If set to N (NO), a normal message will be prepared with the MSGID set. If set to Y (YES), All addresses, message

information and the OPER or EXER set will be printed on cover sheet(s) for review of the transmitting unit. The following page will begin with the UNITID set and contain the unit particular items. The RMKS and DCLAS sets will listed on the last page.

2. LOCAL -MESSAGE INFORMATION- (Figure A-7)

This area is concerned with message addresses, classification, declassification and priority. A maximum of fifteen (15) ACTION and fifteen (15) INFO addressees are processed by the display of Figure A-8. HELP is available to determine ACTION and INFO addressees once ACTION/INFO has been selected. If a message is classified, declassification instructions are required to continue.

3. LOCAL -MISCELLANEOUS- (Figure A-9)

These data sets are recurrent for any UNITREP situation reported.

-1- Operation/Exercise: This creates the OPER or EXER set as required. Only an Operation OR an Exercise can be in progress. This remains fixed until changed by a subsequent UNITREP. NO amplification (AMPN) is allowed.

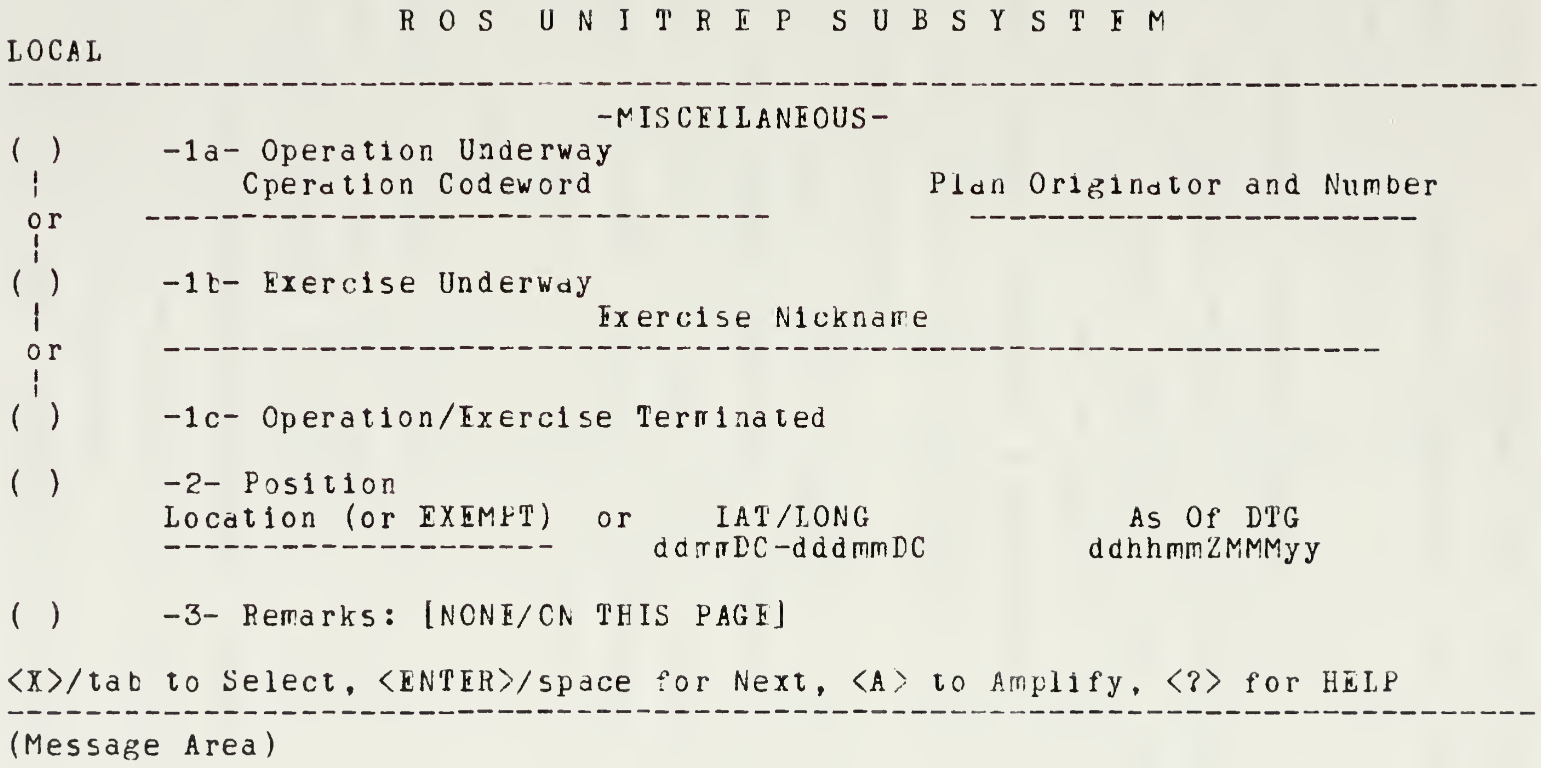
-2- Position: This creates the POSIT set. If the unit is a ship or submarine, the program automatically goes to this field. All other units must select this to indicate a change in position. If the position is 'EXEMPT' no date-time group is required.

```
                   R O S   U N I T R E P   S U B S Y S T E M
LOCAL
-------------------------------------------------------------------------------
                              -MESSAGE INFORMATION-
( ) -1- Addresses
        ( ) Criginator --------------------------------------------------
        ( ) Action
        ( ) Info

( ) -2- Classification                    ( ) -3- Precedence
         - Unclas                                  - Routine
         - Confidential                            - Priority
         - Secret                                  - Immediate
         - Top Secret                              - Flash


( )     -4- Declassification Instructions
           -------------------------------------------------------------


<X>/tat to Select, <ENTER>/space for Next, <?> for HELP
-------------------------------------------------------------------------------
(Message Area)
```

Figure A-7. ROS UNITREP Local Message Display

```
                    R O S   U N I T R E P   S U B S Y S T E M
[ACTION/INFO] ADDRESSEES
------------------------------------------------------------------------------
     The '.' marks the starting point for continuation lines

       -----.--------------------------------------------
       -----.--------------------------------------------
       -----.--------------------------------------------
       -----.--------------------------------------------
       -----.--------------------------------------------
       -----.--------------------------------------------
       -----.--------------------------------------------
       -----.--------------------------------------------
       -----.--------------------------------------------
       -----.--------------------------------------------
       -----.--------------------------------------------
       -----.--------------------------------------------
       -----.--------------------------------------------
       -----.--------------------------------------------
       -----.--------------------------------------------
<ENTER> to Continue, <C> to Change, <E> to Erase, <?> for HELP
------------------------------------------------------------------------------
(Message Area)
```

Figure A-8. ROS UNITREP Message Addressee Display



```
                    R C S   U N I T R E P   S U B S Y S T E M
LOCAL
---------------------------------------------------------------------------
                          -MISCELLANEOUS-
( )     -1a- Operation Underway
            Operation Codeword                 Plan Originator and Number
 ¦
 or     ---------------------------------        -----------------------
 ¦
( )     -1b- Exercise Underway
 ¦                            Exercise Nickname
 or     ----------------------------------------------------------------
 ¦
( )     -1c- Operation/Exercise Terminated

( )     -2- Position
        Location (or EXEMPT)  or    LAT/LONG              As Of DTG
        --------------------     ddmmDC-dddmmDC         ddhhmmZMMMyy

( )     -3- Remarks: [NONE/ON THIS PAGE]

<X>/tab to Select, <ENTER>/space for Next, <A> to Amplify, <?> for HELP
---------------------------------------------------------------------------
(Message Area)
```

Figure A-9. RCS UNITREP Local Miscellaneous Display

-3- REMARKS: Remarks are used to amplify an entire message. A maximum of one hundred (100) lines may be entered using the display of Figure A-5. Ten (10) lines are processed per screen.

E. ADMINISTRATIVE SITUATION (Figure A-10)

This information relates to OPNAVINST 3503.5 section 4, Administrative Status Reporting. Five data sets are provided, all of which may be amplified (AMPN).

-1- Command Change: (COMMAND) Requires the operator to enter the new commander and the effective date-time group of the command change.

-2- Activity Change: (ACTIV) Requires a two letter code for the units current activity as obtained from OPNAVINST 3503.5 Table B-6. This table is available on screen through the HELP facility. The operator input is cross checked against the table to ensure a valid code has been entered.

-3- Physician Status: (MEDIC) Access to this data set is limited to only units which are ships or submarines (OPNAVINST 3503.5 paragraph 4.5). This is a check off item as the possibile entries are limited. The operator enters an 'X' at the appropriate access point. At least one choice must be selected.

-4- New Reporting ORG: (REPORG) The operator checks the appropriate reporting organization and enters the DTG.

```
                    R O S   U N I T R E P   S U B S Y S T E M
ADMINISTRATIVE
------------------------------------------------------------------------------

( )   -1-  Command Change          CO/OIC    ----------------------------------
                                   Effective DTG                  ddhhmmZMMMyy

( )   -2-  Activity Change         Code  --

( )   -3-  Physician Status        ( ) Onboard
                                   ( ) Departed

( )   -4-  New Reporting ORG       Effective DTG                  ddhhmmZMMMyy
           ( )CNO    ( )CINCPACFLT  ( )CINCLANTFLT   ( )CINCUSNAVEUR

( )   -5-  Feedback Verification   ( ) Valid
                                   ( ) Corrected


<X>/tab to Select, <ENTER>/space for Next, <A> to Amplify, <?> for HELP
------------------------------------------------------------------------------
(Message Area)
```

Figure A-10. ROS UNITREP Administrative Display

-5- Feedback Verification: (VERIFY) This set is submitted in reply to a feedback verification request (OPNAVINST 3503.5 paragraph 4.8). The operator selects the appropriate reply dependent on the outcome of the report verification.

C. AIR SITUATION

This information relates to OPNAVINST 3503.5 section 6 - Aircraft and Crews Status Reporting. Certain limitations were imposed in the design of the RCS UNITREP SUBSYSTEM - a unit may possess up to four (4) different aircraft types and deploy these aircraft to no more than four (4) different locations for each type. If a unit possesses less than four (4) different aircraft types, the number of possible locations can be increased by specifying the possessed aircraft type up to the maximum limit of four (4). Each additional specification allows four (4) more locations.

The RCS UNITREP AIR SITUATION is subdivided into three displays: an Authorization Page , a Location Page and a Status Page.

1. AUTHORIZATION (Figure A-11)

This display shows the various Aircraft Types a unit has available, the number of Aircraft and Crews Authorized and the number of Locations where they are deployed. The cursor positions to the ACCESS point of the Type unless the Type is empty. In the latter case entry is to the first blank

```
                    R O S   U N I T R E P   S U B S Y S T E M
AIR AUTHORIZATIONS
--------------------------------------------------------------------------------

              Aircraft           Aircraft        Crews           Deployed
  ACCESS        Type            Authorized     Authorized        Locations
   ( )        -------           ( )     --     ( )     --        ( )     --
   ( )        -------           ( )     --     ( )     --        ( )     --
   ( )        -------           ( )     --     ( )     --        ( )     --
   ( )        -------           ( )     --     ( )     --        ( )     --



--------------------------------------------------------------------------------
<C> to CHANGE, <D> to DELETE, <R> to RESTORE, <A> to AMPLIFY, <?> for HELP
                       *** <X> for Access ***
At ACCESS ( )...............<X> <?> , <C> to change Type (error on entry only)
At an Authorization ( )....<C> <D> <R> <A> <?>
At a Location ( )..........<?>,<X> to view the Locations of the type aircraft

                     <ENTER> for next ( )
--------------------------------------------------------------------------------
(Message Area)
```

Figure A-11. ROS UNITREP Air Authorizations Display

Aircraft Type field. The accessed/entered type is cross checked against OPNAVINST 3503.5 Table 3-2. This table is available to the user through the HELP facility. The cursor next positions to the Authorized Aircraft (AIRAUTH) and Authorized Crews (CREWAUTH) access points. The appropriate access code is entered to change, amplify, delete, restore or request HELP. The cursor positions to the Deployed Location access point. The Location Page is displayed if an 'X' is entered.

2. LOCATION (Figure A-12)

The Location page displays up to four (4) locations at which the aircraft are deployed. The first entry is always named HOME PLATE, a pseudonym for the parent organization location reported in the POSIT set. On message copy it is represented as a hyphen ('-') as required by OPNAVINST 3503.5. The operator enters a new location in the first blank field or selects which location to access for status information.

3. STATUS (Figure A-13)

This page displays the current status of the particular aircraft type at the particular location. All status information for a given location may be deleted by entering a 'D' at the Location access point. Individual status items may be deleted by entering a 'D' at the appropriate access point. Whenever status items are marked for deletetion the operator is prompted to modify a

```
                    R O S   U N I T R E P   S U B S Y S T E M
AIR LOCATION
---------------------------------------------------------------------------
                            Aircraft Type xxxxxxx

        ACCESS                              Location
         ( )                    HOME PLATE-------------------
         ( )                    -----------------------------
         ( )                    -----------------------------
         ( )                    -----------------------------


---------------------------------------------------------------------------

          <X> at ACCESS next to desired location
               (automatic entry into first empty location for NEW)
          <C> to change a location name -- MISPELLING ONLY
          <?> for HELP

          <ENTER> for next ( )
---------------------------------------------------------------------------
(Message Area)
```

Figure A-12. ROS UNITREP Air Location Display

```
                       R O S   U N I T R E P   S U B S Y S T E M
AIR STATUS
--------------------------------------------------------------------------------
Aircraft Type xxxxxxx           ( )   Location  xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

    ( )      -1- Aircraft                            ( )      -2- Crews
 Possessed     FMC   PMC   NMC                       Formed        Ready
    --         --    --    --                          --           --

      ( )    -3- Reconnaissance Capability  Primary --------

        --------   --------   --------   --------   --------
        --------   --------   --------   --------   --------
--------------------------------------------------------------------------------
ENTER   <C> - to CHANGE -  (in fields 1,2,3 only)
        <D> - to DELETE -  (at Location will DELETE ALL information)
        <R> - to RESTORE - (restores deleted fields)
        <A> - to AMPLIFY - (in fields 1,2,3 only)
        <?> - for HELP -   (all fields)
THEN: ENTER Required information.
       (an <ENTER> with no information is assummed to be NO CHANGE)
--------------------------------------------------------------------------------
(Message Area)
```

Figure A-13. RCS UNITREP Air Status Display

corresponding status display, if required (e.g. Deletion of a detachment's status information normally requires an increase in the parent organization's number of aircraft and crews possessed).

-1- Aircraft: (AIRSTAT) Full Mission Capable/Partial Mission Capable (FMC/PMC) are always reported within the AIRSTAT set. A check is made to insure that Aircraft Possessed = FMC + PMC + NMC (Not Mission Capable). If this set is deleted the Reconaissance Capability is assumed to have been deleted also. Therefore RECON will NOT be reported as deleted, but will in fact be purged from the system (OPNAVINST 3503.5 section 6 paragraph 8e).

-2- Crews: (CREWSTAT) Crews Formed are not allowed to exceed Crews Ready.

-3- Reconnaissance Capability: (RECON) As this set is reported in its entirety whenever a change has been made, individual capability deletions and additions are permitted. The operator is provided a prompt as to the procedures to be followed. The primary recon capability is always entered first. Entries are checked against OPNAVINST 3503.5 Table B-3. The table is also available on screen through the HELP facility.

Enclosure A-1

Sample ROS UNITREP Draft Document

ROS UNITREP DRAFT DATA
EC-1 DET 3   SERIAL 345

CLASSIFY IN ACCORDANCE WITH OPNAVINST 5510.1

```
****** INITIAL ITEMS - DATA SET INFORMATION ******

UNITREP SITUATION  - <X> appropriate items

 (X) -0- Modify Local Information

  ( ) -1-  Personnel Status
  ( ) -2-  Administrative Status
  ( ) -3-  Unit Combat Readiness Assesment
  ( ) -4-  Aircraft and Crews Status
  ( ) -5-  Major Equipment Status
  ( ) -6-  Special Capabilities Status
  ( ) -7-  Increased Defense Readiness Status

  ( ) -8-  Reserve Augmentation Status
  ( ) -9-  Type Commander Reports



   INFREQUENTLY CHANGED ITEMS -
      CHANGE ONLY ON INITIALIZATION AND WHEN NECESSARY

( ) -1- Unit Type:
        ( )Ship   ( )Submarine   ( )Air   ( )Shore   ( )Other


( ) -2- Unit Identification - OPNAVINST 3503.1 App. C
        ----------- --------------------

( ) -3- Last Serial ___


( )   -4- Message to be Prepared as Feeder (Y/N): _
               (UNITID vice MSGID data set)

  If Feeder is set to YES  precedence,addresses and
    classification will be placed on the message cover
    sheet(s) for review by the transmitting unit. The
    UNITID set will be used on the first line of the next
    page. Remarks and declassification lines will be listed
   on the final page.
  If Feeder is set to NO all message information will occur
    in sequence with the MSGID set.

AMPN MSGID/UNITID (Circle if Attached)
```

```
P
FM      HELSUPPRON ONE DET THREE
TO      CINCPACFLT PEARL HARBOR HI
        COMTHIRDFLT
        COMASWWINGPAC SAN DIEGO CA
        NAVIATACEN PEARL HARBOR HI
            ZEN/HELSUPPRON ONE
INFO    COMCARAIRWING FOURTEEN
C O N F I D E N T I A L
       ( Classified for Demonstration Purposes Only )
****** MESSAGE HEADER - DATA SET INFORMATION ******
( ) -1- Addresses
        ( ) Originator

  -------------------------------------------------
        ( ) Action
        ( ) Info

( ) -2- Classification                  ( ) -3- Precedence
         _ Unclas                                _ Routine
         _ Confidential                          _ Priority
         _ Secret                                _ Immediate
         _ Top Secret                            _ Flash


( ) -4- Declassification Instructions
--------------------------------------------------------------

Additional Addressees (Indicate ACTION or INFO)

    -----.-------- --------------- ------------------
    -----.-------- --------------- ------------------
    -----.-------------------------------------------
    -----.-------- --------------- ------------------
    -----.-------------------------------------------
    -----.-------------------------------------------
    -----.-------------------------------------------
    -----.-------------------------------------------
    -----.-------------------------------------------
    -----.-------- ----------------------------------
```

```
EXER/SUMMER RAIN//

****** OPER/EXER - DATA SET INFORMATION ******
  Only an Operation OR an Exercise can be underway, NOT
  BOTH. To change from an Operation to an Exercise or
  vice-verse, first TERMINATE the current Operation or
  Exercise, then initiate  the new Operation/Exercise.

( ) -1a- Operation Underway
 |          Operation Codeword
 |   --------------------------------  (MANDATORY)
 |
       Plan Originator and Number
 or      -----------------------  (OPTIONAL)
 |
( ) -1b- Exercise Underway
 |           Exercise Nickname (62 characters max)
 or      ----------------------------------------------------
         ---------------
 |       (MANDATORY)
( ) -1c- Operation/Exercise Terminated


MSGID/UNITREP/HC-1 DET 3/345//

****** MSGID/UNITID - DATA SET INFORMATION ******
  This data set is set within the program depending on the
  items noted on the cover page under Feeder.

The format is:

MSGID/UNITREP/---message originator---/serial//
   or
UNITID/ ---message originator--- /serial//

AMPN (Circle if Attached)


POSIT/NORTH ISLAND/221100ZOCT83//

****** POSIT - DATA SET INFORMATION ******
  MANDATORY for ships/submarines. IF Location is EXEMPT then
  no other information is required. IF not at a standard
  location, LAT/LONG is required. The DTG is required for
  any Location or LAT/LONG when not exempt.

( ) -2- Position - Required for ship/sub
        Location (or EXEMPT)  or    LAT/LONG
        --------------------      ddmmDC-dddmmDC
```

AS-CF Date-Time Group ------------- ddhhmmZMMMyy
------Z-----

AMPN (Circle if Attached)

( ) -3- Remarks:

Remarks (RMKS) are used to amplify an entire message. They are limited to 100 lines at most.

(Additional attached)

```
COMMAND/LCDR B.F. NEWGUY/211200ZMAR83//

****** COMMAND - DATA SET INFORMATION ******
  BOTH items MANDATORY. Punctation optional

( )   -1-  Command Change

      CO/OIC      ________________________________

      Effective DTG                  ddhhmmZMMMyy
                                     ______Z_____
AMPN (Circle if Attached)


ACTIV/CS//

CS ops - Combat Support


****** ACTIV - DATA SET INFORMATION ******
  Valid codes are contained in TABLE B-6.

( )   -2-  Activity Change          Code  __

AMPN (Circle if Attached)


REPORG/CINCPACFLT/201900ZMAR83//

****** REPORG - DATA SET INFORMATION ******
  Reports the NEW reporting organization

( )   -4-  New Reporting ORG     Effective DTG  ddhhmmZMMMyy
                                                ______Z_____
   ( )CNO    ( )CINCPACFLT    ( )CINCLANTFLT    ( )CINCUSNAVEUR

AMPN (Circle if Attached)


VERIFY/VALID//

****** VERIFY - DATA SET INFORMATION ******
  VALID - all data in the NRS feedback report is CORRECT
  CORRECTED - the appropriate information has been submitted

( )   -5-  Feedback Verification    ( ) Valid
                                    ( ) Corrected
AMPN (Circle if Attached)
```

```
AIRAUTH/SH-3G/AUTH:04//
CREWAUTH/SH-3G/AUTH:04//

****** AIR AUTHORIZATIONS - DATA SET INFORMATION ******
   Aircraft Authorized are listed in the Weapon System
     Planning Document OPNAVNOTE 13210.
   Crews Authorized are listed in the Unit Manpower
     Authorization OPNAVNOTE 1000/2, M+1 column.

                  Aircraft            Crews
    ACCESS       Authorized        Authorized

     ( )           ( )       __       ( )        __
AMPN-AIRAUTH (Circle if Attached)
AMPN-CREWAUTH



AIRSTAT/SH-3G/POSS:04/-/FMC:02/PMC:01//
CREWSTAT/SH-3G/FORM:06/-/READY:05//
RECON/SH-3G/-/CAMERA/TV//

****** Recon Capability Definitions ******
CAMERA: Hand Held Camera
TV: Television

****** AIR STATUS - DATA SET INFORMATION ******
   Valid RECON capabilities are contained in TABLE B-3.

      ( )      -1- Aircraft                 ( )     -2- Crews
  Possessed     FMC    PMC    NMC          Formed       Ready

     --         --     --     --             --           --

      ( )      -3- Reconaissance Capability  Primary ________


        --------   --------   ---- ---   --------   --------
        --------   --------   -- - ---   --------   --------

AMPN-AIRSTAT (Circle if Attached)
AMPN-CREWSTAT
AMPN-RECON



DELETE/AIRSTAT/SH-3G/POSS:02/CORAL SEA/FMC:02/PMC:00//
DELETE/CREWSTAT/SH-3G/FORM:03/CORAL SEA/READY:03//

****** Recon Capability Definitions ******
CAMERA: Hand Held Camera
TV: Television
```

```
****** AIR STATUS - DATA SET INFORMATION ******
   Valid RECON capabilities are contained in TABLE B-3.

     ( )     -1- Aircraft                     ( )     -2- Crews
  Possessed    FMC   PMC   NMC                Formed      Ready

     --         --    --    --                   --          --

     ( )     -3- Reconaissance Capability  Primary ________


       --------   --------   ---- ---   --------   --------
       --------   --------   --------   --------   --------

AMPN-AIRSTAT (Circle if Attached)
AMPN-CREWSTAT
AMPN-RECON
```

```
****** AIR - NEW - DATA SET INFORMATION ******
AUTHORIZATIONS
   Valid Aircraft Types are contained in TABLE B-2.
   Aircraft Authorized are listed in the Weapon System
     Planning Document OPNAVNOTE 13010.
   Crews Authorized are listed in the Unit Manpower
     Authorization OPNAVNOTE 1000/2, M+1 column.

                  Aircraft          Aircraft            Crews
#   ACCESS          Type            Authorized          Authorized

    ( )           -------           ( )     --          ( )     --

LOCATION
    ( )  Location  ------------------------------

STATUS
   Valid RECON capabilities are contained in TABLE B-3.

      ( )      -1- Aircraft                      ( )      -2- Crews
  Possessed     FMC    PMC    NMC                Formed        Ready

      --         --     --     --                  --            --

      ( )      -3- Reconaissance Capability  Primary --------

          --------   --------   --------   --------   --------
          --------   --------   --------   --------   --------

AMPN-AIRAUTH (Circle if Attached)
AMPN-CREWAUTH
AMPN-AIRSTAT
AMPN-CREWSTAT
AMPN-RECON
```

RMKS/LIFE GUARD DETACEMENT RETURNED FROM SHORT CRUISE//

****** RMKS - DATA SET INFORMATION ******
Change above. Item -3- Remarks.

DCLAS/DECL 21 MAR 87//

****** DCLAS - DATA SET INFORMATION ******
The declassification line is set in accordance with OPNAVINST 5510.1. It is required for all classified messages and is set on the cover page under Message Header item -4-.

P

FM HELSUPPRON ONE DET THREE
TO CINCPACFLT PEARL HARBOR HI
COMTHIRDFLT
COMASWWINGPAC SAN DIEGO CA
NAVDATACEN PEARL HARBOR HI
ZEN/HELSUPPRON ONE
INFO COMCARAIRWING FOURTEEN

CONFIDENTIAL
(Classified for Demonstration Purposes Only)
EXER/SUMMER RAIN//

MSGID/UNITREP/HC-1 DET 3/345//

COMMAND/LCDR B.F. NEWGUY/211200ZMAR83//

AIRSTAT/SH-3G/POSS:04/-/FMC:02/PMC:01//

CREWSTAT/SH-3G/FORM:06/-/READY:05//

DELETE/AIRSTAT/SH-3G/POSS:02/CORAL SEA/FMC:02/PMC:00//

DELETE/CREWSTAT/SH-3G/FORM:03/CORAL SEA/READY:03//

RMKS/LIFE GUARD DETACHMENT RETURNED FROM SHORT CRUISE//

DCLAS/DECL 21 MAR 87//

NNNN

APPENDIX B

JANUS/ADA LIBRARY MODULES

This appendix provides the definitions of the subprograms from the JANUS/ADA library packages (io, util, strlib, chainlib) used in the RCS UNITREP SUBSYSTEM.

```
Package IO Is

-- The I/O package for JANUS V. 1.4.5
-- Last Modified  10/13/82


    -- Copyright 1982  RR Software, P.O. Box 1512, Madison
    -- WI  53701.  Permission is hereby given to  distribute
    -- Object Code produced from these libraries.


Type File_Mode Is
               (No_Access,Read_Only,Write_Only,Read_Write);

IOresult : Integer; -- The result of the IO operation

Procedure Open(Fyle : In Out File; Name : In String;
                                   Mode : In File_Mode);
    -- Open the file name and give it the mode mode

Procedure Create(Fyle : In Out File; Name : In String;
                                     Mode : In File_Mode);
    -- Create the file name and give it the mode mode

Procedure Delete(Name : In String);
    -- Delete the file name

Procedure Close(Fyle : In Out File);
    -- Close the file fyle

Function Name(Fyle : In File) Return String;
    -- Return the name of the Open file

Function Is_open(Fyle : In File) Return Boolean;
    -- Is the file fyle open?
```

```
Function Get_Line(Fyle : In File) Return String;
    -- Get a line from the file fyle

Procedure Put_Hex(Fyle : In File; val : In Integer);
    -- Write the integer in hexidecimal (no special format)

Function End_of_file(fyle : In File) Return Boolean;
    -- End of File Reached (in a text file)?

Function EOF(fyle : In File) Return Boolean;
    -- End of File Reached (in a binary file)?

Function Disk_full(fyle : In File) Return Boolean;
    -- Is the Disk full ?

Function End_of_Line(fyle : In File) Return Boolean;
    -- End of Line Reached?

End IO;
```

```
With Jlib80;
Package Util Is
    -- Spec for the package util
    -- Last modifed 9/22/82
    -- Contains the utility routines, and the basic file
    -- handling routines

    -- Copyright 1982  RR Software, P.O. Box 1512, Madison
    -- WI 53701. Permission is hereby given to distribute
    -- Object Code produced from these libraries.


    Use JLib80; -- So the file definitions are available

    Procedure Err_Exit;
    Procedure Halt;
    Function Hi (val : Integer) Return Byte;
    Function Lo (val : Integer) Return Byte;
    Function Memavail Return Integer;
    Function Maxavail Return Integer;
    Function Command_Line Return String;
        -- Returns the command line

    -- Default File Procedures

    Function FConvert(Fyle : In File) Return File_ptr;
    Procedure FFConvert(Fyle_ptr : In File_Ptr;
                        Fyle : Out File);
        -- Convert  to  and from the type file to  the  type
        -- file_ptr.  For system use only, Not to be used in
        -- user programs.

    Function Standard_Input Return File;
        -- Returns the initial default system input file

    Function Standard_Output Return File;
        -- Returns the initial default system output file

    Function Current_Input Return File;
        -- Returns the current default input file

    Function Current_Output Return File;
        -- Returns the current default output file

    Procedure Set_Input(Fyle : In File);
        -- Set the current default input file to fyle

    Procedure Set_Output(Fyle : In File);
        -- Set the current default output file to fyle

End Util;
```

```
Package Strlib Is
    -- String Handling Package Specification
    -- Last Modified 6/ 3/82


    -- Copyright 1982  RR Software,  P.C.  Box 1512, Madison
    -- WI  53701.  Permission is hereby given to  distribute
    -- Object Code produced from these libraries.


    Subtype Mstring Is String(255);
                -- Maximum string length
    Subtype StrIndex Is Integer Range 0..255;
                -- Maximum string indices

Function Length (str : In Mstring) Return Integer;
    -- Return the length of the string

Function Remove (str : In Mstring; pos,size : In StrIndex)
                                                   Return Mstring;
    -- Remove size characters from str at pos

Function Insert (source,dest : In MString;
                           pos : In StrIndex) Return MString;
    -- Insert source into dest at pos

Function Extract (str : In Mstring; pos,size : In StrIndex)
                                                   Return Mstring;
    -- Extract size characters from str at pos

Function Position (pattern,str : Mstring) Return Integer;
    -- Return the position of the first occurance of pattern
    -- in str, or 0 if there is none

Function char_to_str (char : character) Return String;
    -- Convert a character into a string of length 1

Function str_to_int (str : Mstring) Return Integer;
    -- Convert a string into an integer

Function int_to_str (int : Integer) Return Mstring;
    -- Convert an integer into a string

End Strlib;
```

```
Package Chainlib Is
    -- The program chaining and calling library
    -- Last modified 9/ 9/82

    -- Copyright 1982  RR Software,  P.O.  Box 1512, Madison
    -- WI  53701.  Permission is hereby given to  distribute
    -- Object Code produced from these libraries.

    Procedure Chain(Str : In String);
        -- Chains a program, saving the data segment
        -- Note:  The  Jlib80  library must be the same  for
        -- both  the chaining and chained programs for  this
        -- routine to work

    Procedure Simple_Chain(Str : In String);
        -- Chains a program, destroying the data segment

    Procedure Prog_Call(Str : In String);
        -- Calls a program (Not Implemented)

    Procedure Prog_Return;
        -- Returns from a called program (Not Implemented)

End Chainlib;
```

APPENDIX C

ROS UNITREP COMPUTER LISTING

This appendix provides the computer listing of the ROS UNITREP SUBSYSTEM. The listing is organized to follow the structure of Chapter IV (Implementation):

Basic Program Support Components
- consio
- printio
- urglobal
- urutil
- urtest

Program Data Components
- urglbl
- urlocalA
- uradminB
- urairB
- unitrepA
- unitrepB

Program Control Components
- unitrep
- unitrep1
- unitrep2
- unitrep3
- filerA
- filerB

Program Operative Components
- urlocal
- urlocal1
- uradmin
- urair
- urair1
- urair2
- urair3

Miscellaneous
- initialA
- initialB

```
PACKAGE consio IS
        -- This package provides console screen oriented
        -- i/o procedures for a Datamedia Elite 2500
        -- terminal

  bell: CONSTANT := CHARACTER'VAL(7);    -- Aural beep
  carriage_return: CONSTANT := CHARACTER'VAL(13);
  tab: CONSTANT := CHARACTER'VAL(9);

  PROCEDURE getxy_immediate(x,y: IN INTEGER;
                            char: OUT CHARACTER);
        -- Positions the cursor at screen position x,y and
        -- waits for exactly one character to be input.
        -- Converts to BAUDOT/OCR only characters

  PROCEDURE getxy(x,y: IN INTEGER;
                  temp: OUT STRING;
                  field_length: IN INTEGER);
        -- This procedure positions the cursor at screen
        -- position x,y reads the input, temp  for a maximum
        -- of field_length characters, and converts to
        -- BAUDOT/OCR characters if required.

  PROCEDURE putxy(x,y: IN INTEGER; temp: IN STRING);
        -- Positions the cursor at screen position x,y and
        -- writes the input, temp, at this position.

  PROCEDURE clear_screen;
        -- Clears the console display.

  FUNCTION mark(xx,yy: IN INTEGER) RETURN BOOLEAN;
        -- Checks for an <X>|tab or carriage return|space
        -- at the x/y coordinates

  PROCEDURE clear_field(x,y,len: IN INTEGER);
        -- Clears the screen field of length len at x,y

END consio;
```

```
  WITH strlib,io,util; -- JANUS library packages
PACKAGE BODY consio IS
        -- This package provides console screen oriented
        -- i/o procedures for Datamedia Elite 2500.
        -- Must be tailored to the particular system.

        -- terminal dependent constants
  position_cursor: CONSTANT := CHARACTER'VAL(12);
  clear: CONSTANT := CHARACTER'VAL(30);
  rubout: CONSTANT := CHARACTER'VAL(127);
  screen_backspace: CONSTANT := CHARACTER'VAL(8);

  PROCEDURE gotoxy(x,y: IN INTEGER) IS
        -- This procedure is modified for the screen of a
        -- DATAMEDIA ELITE 2500 terminal. It must be changed
        -- as required for each new terminal
    xx,yy: INTEGER;
    BEGIN
    CASE x IS
      WHEN 0..31 => xx := x +96;
      WHEN 32..63 => xx := x + 32;
      WHEN 64..78 => xx := x - 32;
      WHEN OTHERS => PUT("Invalid x coordinate");
                     xx := 96;
    END CASE;

    CASE y IS
      WHEN 0..22 => yy := y + 96;
      WHEN OTHERS => PUT("Invalid y coordinate");
                     yy := 96;
    END CASE;
        -- set addressing mode and coordinates for xy
        -- terminal addressing
    PUT(position_cursor);
    PUT(CHARACTER'VAL(xx));
    PUT(CHARACTER'VAL(yy));
  END gotoxy;

  PROCEDURE baudot_convert( char: IN OUT CHARACTER) IS
    USE util;   -- Halt
        -- Takes char and converts to legal BAUDOT/OCR
        -- character. If lower case, capitalizes. If
        -- illegal converts to null and puts out an aural
        -- tone
    BEGIN
      CASE char IS
                -- no action on these control characters
        WHEN CHARACTER'VAL(0) | CHARACTER'VAL(2) |
             CHARACTER'VAL(4)..CHARACTER'VAL(8)|
             CHARACTER'VAL(10)..CHARACTER'VAL(31)|
             CHARACTER'VAL(127)                   => NULL;
```

```
              -- abort and reboot on control-c
      WHEN CHARACTER'VAL(3) => clear_screen;
           PUT("User Commanded Abort ...");
           Halt;
              -- allow tab 'VAL(9) to function as a mark
      WHEN CHARACTER'VAL(9) => char := 'X';
              -- allowable BAUDOT characters
      WHEN '0'..'9' | 'A'..'Z' | '-' | '?' | ':' |
         '$' | '&' | '#' | ''' | '(' | ')' | '.' |
         ',' | ';' | '/' | '"' | ' '                => NULL;
              -- capitalize lower case
      WHEN 'a'..'z' =>
        char := CHARACTER'VAL( CHARACTER'POS(char) -32);
              -- convert all others to null and notify
              -- user by the bell
      WHEN OTHERS => char := CHARACTER'VAL(0); PUT(bell);
    END CASE;
  END baudot_convert;

PROCEDURE getxy_immediate(x,y: IN INTEGER;
                          char: OUT CHARACTER) IS
  USE io;     -- Open,Close
      -- gets exactly one character from screen position
      -- x,y and converts to BAUDOT/OCR
  fyle: FILE;
  BEGIN
    Open(fyle,"KBD:",Read_Only);      -- JANUS keyboard
    gotoxy(x,y);
    GET(fyle,char);
    CLOSE(fyle);
    baudot_convert(char);
    PUT(char);
  END getxy_immediate;


PROCEDURE getxy(x,y: IN INTEGER;
                temp: OUT STRING;
                field_length: IN INTEGER) IS
  USE strlib;           -- Length,Extract,Char_to Str
      -- This procedure positions the cursor at screen
      -- position x,y reads the input, temp for a maximum
      -- of field_length characters, and converts to
      -- BAUDOTOCR characters if required. Manipulation
      -- for character deletion allowed.

  key: CHARACTER;
  xnext: INTEGER := x;
  BEGIN
    temp := "";
    LOOP
      getxy_immediate(xnext,y,key);
```

```
      EXIT WHEN key = carriage_return;  -- no more input
      IF key = rubout AND xnext > x THEN
        PUT(screen_backspace);
        PUT('-');     -- erase screen character
        temp := Extract(temp,1,Length(temp)-1);
              --remove from string
        xnext := xnext-1;
      ELSIF key = rubout and xnext = x THEN
        PUT(bell);
      ELSIF key = CHARACTER'VAL(0) THEN
        NULL; -- illegal character
      ELSE
        temp := temp & Char_to_Str(key);
        xnext := xnext+1;
        EXIT WHEN xnext > field_length + x - 1;
      END IF;
    END LOOP;
              -- strip trailing blanks
    FOR i IN REVERSE 1..Length(temp) LOOP
      EXIT WHEN temp(i) /= ' ';
      IF temp(i) = ' ' THEN
        temp := Extract(temp,1,i-1);
      END IF;
    END LOOP;
  END getxy;


PROCEDURE putxy(x,y: IN INTEGER; temp: IN STRING) IS
      -- Positions the cursor at screen position x,y and
      -- writes the input, temp, at this position.
  BEGIN
    gotoxy(x,y);
    PUT(temp);
  END putxy;

PROCEDURE clear_screen IS
      -- Clears the console screen
  BEGIN
    PUT(clear);
  END clear_screen;


FUNCTION mark(xx,yy: IN INTEGER) RETURN BOOLEAN IS
      -- used in menu selection areas where a boolean
      -- value is required by the program.
  key: CHARACTER;
  BEGIN
    LOOP
      getxy_immediate(xx,yy,key);
      IF key = 'X' THEN
        RETURN TRUE;
```

```
        ELSIF key = tab THEN
          RETURN TRUE;
        ELSIF key = carriage_return THEN
          RETURN FALSE;
        ELSIF key = ' ' THEN
          RETURN FALSE;
        ELSE
          PUT(bell);
        END IF;
      END LOOP;
    END mark;

  PROCEDURE clear_field(x,y,len: IN INTEGER) IS
        -- clears screen data fields for len characters
    BEGIN
      FOR i IN 0..len-1 LOOP
        putxy(x+i,y,"-");
      END LOOP;
    END clear_field;

END consio;
```

```
PACKAGE printio IS
        -- Provides interface to the printer

  line_count: INTEGER;

        -- format parameters
  max_lines_in_page,max_line_length,
          line_spacing,header_lines:INTEGER;

  margin,end_trans: STRING;

  printer: FILE;

  PROCEDURE printer_on;
  PROCEDURE printer_off;

  PROCEDURE NEW_PAGE;
        -- page eject

  TYPE format IS (msg,ocr,text);
  TYPE format_state IS ARRAY
                      (format RANGE msg..text) OF BOOLEAN;
  format_is: format_state;

  PROCEDURE set_format(format_type: IN format);
        -- sets page formatting

  PROCEDURE put_printer(content: IN STRING);
        -- prints the content

END printic;
```

```
  WITH strlib,io;       --JANUS/ADA libraries
PACKAGE BODY printio IS
        -- Provides interface to the printer

                -- for Teletype Corp model 40 printer
  page_eject: CONSTANT := CHARACTER'VAL(12);

  PROCEDURE printer_on IS
    USE io;     -- Open
                -- JANUS/ADA printer is LST:
    BEGIN
      Open(printer,"LST:",Write_Only);
    END printer_on;

  PROCEDURE printer_off IS
    USE io;     -- CLOSE
    BEGIN
      CLOSE(printer);
    END printer_off;

  PROCEDURE NEW_PAGE IS
        -- page eject. Can be eliminated with ADA
        -- supporting the NEW_PAGE procedure
    BEGIN
      PUT(printer,page_eject);
      line_count := 1;
    END NEW_PAGE;


  PROCEDURE set_format(format_type: IN format) IS
        -- sets page formatting
    BEGIN
      format_is(msg) := FALSE;
      format_is(ocr) := FALSE;
      format_is(text) := FALSE;
      IF format_type = msg THEN
        format_is(msg) := TRUE;
        max_lines_in_page := 42;
        max_line_length := 69;
        line_spacing := 1;
        header_lines := 9;
        margin := "       ";
        end_trans := "BT";
      ELSIF format_type = ocr THEN
        format_is(ocr) := TRUE;
        max_lines_in_page := 19;
        max_line_length := 69;
        line_spacing := 2;
        header_lines := 6;
        margin := "      ";
        end_trans := "NNNN";
```

```
      ELSE      -- format_type = text
        format_is(text) := TRUE;
        max_lines_in_page := 55;
        max_line_length := 80;
        line_spacing := 1;
        header_lines := 6;
        margin := "";
        end_trans := "";
      END IF;
    END set_format;


  PROCEDURE put_printer(content: IN STRING) IS
    USE strlib;          -- Length,Extract
        -- prints the content
    buffer: STRING;
    BEGIN
      buffer := content;
        -- limit line length according to format
      IF Length(buffer) > max_line_length THEN
        buffer := Extract(buffer,1,max_line_length);
      END IF;
      buffer := margin & buffer;
        -- print header
      IF line_count = 1 THEN
        FOR i IN 1..header_lines LOOP
          NEW_LINE(printer);
        END LOOP;
      END IF;
        -- print line
      PUT(printer,buffer);
        -- line spacing
      FOR i IN 1..line_spacing LOOP
        NEW_LINE(printer);
      END LOOP;
      line_count := line_count + 1;
        -- page_eject
      IF line_count > max_lines_in_page THEN
        NEW_PAGE;
      END IF;
    END put_printer;

END printio;
```

```
  WITH printio,
       consio,
       urgltl;
PACKAGE urglobal IS
  USE printio,             -- for format in program status
      consio,              -- for carriage_return
      urgltl;              -- for comment_set/process_comment

        -- global variables

  fyle: FILE;

  key: CHARACTER;                  -- single key read

                -- constant single Characters
  x: CONSTANT := ´X´; y: CONSTANT := ´Y´;
  n: CONSTANT := ´N´; h: CONSTANT := ´H´;
  c: CONSTANT := ´C´; e: CONSTANT := ´E´;
  a: CONSTANT := ´A´; d: CONSTANT := ´D´;
  r: CONSTANT := ´R´;
  space: CONSTANT := ´ ´; ques: CONSTANT := ´?´;
  no_input: CONSTANT := carriage_return;

  file_buffer,buffer: STRING;      -- file/keyboard buffers
  blankln,dashln: STRING(80);      -- screen lines

        -- strings of length 1
  dash,blank,d_mark,n_mark,r_mark,x_mark,y_mark,
   a_mark,star,null_string:  STRING;

  success,reselect: BOOLEAN;       -- general booleans
  number: INTEGER;                 -- general numbers

                -- global action data

  TYPE action IS (draft,neww,modify,print,
                                      log,erase,verify,quit);
  TYPE action_state IS ARRAY (action RANGE draft..quit)
                                                  OF BOOLEAN;
  action_is: action_state;

                -- global areas

  TYPE areas IS (pers,admin,combat,air,equip,specap,
                              defense,reserve,tycom,local);
  TYPE area_state IS ARRAY (areas RANGE pers..local)
                                                  OF BOOLEAN;
  area_is: area_state;

  TYPE unit_type IS (ship,submarine,air_unit,shore,other);
```

```
        -- program status structure used to communicate
        -- between the program modules
  TYPE program_status IS
    RECORD
      unit: unit_type;
      initial_entry: BOOLEAN;
      print_trailer: BOOLEAN;
      message_transmitted: BOOLEAN;
      current_action: action_state;
      current_area: area_state;
      current_format: format_state;
      printing_line: INTEGER;
      workfile: STRING(14);
    END RECORD;

  program:program_status;


  PROCEDURE border;
        -- screen outline

  PROCEDURE process_comment(data_set: IN STRING;
                            comment: IN OUT comment_set);
        -- processes free text (AMPN/RMKS) data sets

  PROCEDURE print_comment(data_set: IN STRING);
        -- prints amplification (AMPN) data sets

  PROCEDURE error(number: IN INTEGER);
        -- Provides console error messages

  PROCEDURE help(data_set: IN STRING);
        -- Provides RCS UNITREP HELP information

  PROCEDURE view_table(filename: IN STRING);
        -- Provides user with the option to examine valid
        -- table entries from OPNAVINST 3503.5

  FUNCTION required_print(changed_item: BOOLEAN)
                                              RETURN BOOLEAN;
        -- determines if a given data set is to be printed
        -- for the required report

  PROCEDURE draft_aid(data_set: IN STRING);
        -- provides assistance in form of a draft document
        -- gets the draft information for data_set

END urglobal;
```

```
  WITH io,strlib,          -- JANUS/ADA libraries
       consio,printio,
--#       urtest,               -- in procedure error/help
       urgltl;
PACKAGE BODY urglobal IS
  USE consio,
      urgltl;
        -- global variables initialization performed here as
        -- JANUS/ADA cannot initialize structured variables.
        -- Would be accomplished in specification in full
        -- ADA.

  PROCEDURE border IS
        -- draws the standard border
    BEGIN
      clear_screen;
      putxy(20,0,
       "R C S   U N I T R E P   S U B S Y S T E M");
      putxy(0,2,dashln);
      putxy(0,21,dashln);
    END border;

  PROCEDURE process_comment(data_set: IN STRING;
                            comment: IN OUT comment_set) IS
    USE strlib;          --Extract
    USE io;      --Open,CLOSE,Create,Delete,GET_LINE

        -- processes all comment sets 10 lines/time

    content: ARRAY (1..max_comment) OF
                                       STRING(max_line_length);
    slo: CONSTANT := 5;          --screen line offset

    PROCEDURE commentmask IS
      USE strlib;        -- Extract
        -- draws the comment screen
      BEGIN
        border;
        IF Extract(data_set,1,4) = "RMKS" THEN
          putxy(0,3,"REMARKS SECTION - PAGE "
                & Extract(data_set,5,1));
        ELSE             -- comment is remarks
          putxy(0,3,"AMPLIFICATION SECTION - " & data_set);
        END IF;
        FOR line_number IN  1..max_comment LOOP
          putxy(5,line_number + slo,
                Extract(dashln,1,max_line_length));
        END LOOP;
      END commentmask;
```

```
PROCEDURE fill_commentmask IS
  BEGIN
    commentmask;
    FOR line_number IN 1..comment.number_of_lines LOOP
      putxy(5,line_number + slo,content(line_number));
    END LOOP;
  END fill_commentmask;

BEGIN
            -- read in existing data
  IF comment.change THEN
    Open(fyle,data_set  & ".AMP",Read_Only);
    FOR line_number IN 1..comment.number_of_lines LOOP
      content(line_number) := GET_LINE(fyle);
    END LOOP;
    FOR line_number IN comment.number_of_lines+1..
                                     max_comment LOOP
      content(line_number) := null_string;
    END LOOP;
    CLOSE(fyle);
  END IF;
            -- process comment data
  IF NOT comment.change THEN
    comment.number_of_lines := 0;
    commentmask;
    FOR line_number IN 1..max_comment LOOP
      content(line_number) := null_string;
    END LOOP;
    FOR line_number IN 1..max_comment LOOP
      getxy_immediate(5,line_number+slo,key);
      EXIT WHEN key = no_input OR key = space;
      comment.change := TRUE;
      getxy(6,line_number + slo,content(line_number),61);
      content(line_number) := Char_to_Str(key)
                              & content(line_number);
      comment.number_of_lines := line_number;
    END LOOP;
  END IF;
  comment_page: LOOP
    fill_commentmask;
    putxy(2,20,"<ENTER> to continue, <C> to change, " &
              "<E> to erase, <?> for help");
    getxy_immediate(78,20,key);
    CASE key IS
      WHEN no_input | space => EXIT;
      WHEN c =>
        putxy(0,20,blankln);
        putxy(0,20,"Position cursor to line to change "
        &"and type over - ENTIRE LINE MUST BE CHANGED");
        FOR line_number IN 1.. max_comment LOOP
          getxy_immediate(5,line_number + slo,key);
```

```
            IF key = no_input OR key = space THEN
              putxy(5,line_number+slo,
                                    content(line_number));
            ELSE
              clear_field(6,line_number + slo,61);
              getxy(6,line_number + slo,buffer,61);
              content(line_number) := Char_to_Str(key)
                                      & buffer;
              IF line_number > comment.number_of_lines
                                                      THEN
                comment.number_of_lines := line_number;
              END IF;
            END IF;
          END LOOP;
        WHEN e => comment.change := FALSE;
          FOR line_number IN 1..max_comment LOOP
            content(line_number) := null_string;
          END LOOP;
        WHEN ques => help("AMPN/RMKS");
                     fill_commentmask;
        WHEN OTHERS => error(6);        -- incorrect input
      END CASE;
      IF content(1) = null_string THEN
        comment.change := FALSE;
        comment.number_of_lines := 0;
      END IF;
    END LOOP comment_page;
              -- Write data to file if required
    DELETE(data_set & ".AMP");
    IF comment.change THEN
      Create(fyle,data_set & ".AMP",Write_Only);
      FOR line_number IN 1..comment.number_of_lines LOOP
        PUT(fyle,content(line_number));
        NEW_LINE(fyle);
      END LOOP;
      CLOSE(fyle);
    END IF;
  END process_comment;

PROCEDURE print_comment(data_set: IN STRING) IS
  USE printio;          --put_printer
  USE io;      --Open,GET_LINE,END_OF_FILE,CLOSE
      -- print any amplification/remarks lines required
      -- for old messages - .amp files are purged
      -- when message is logged as transmitted
  BEGIN
      Open(fyle,data_set & ".AMP",Read_Only);
      file_buffer := "AMPN/" & GET_LINE(fyle);
      LOOP
        IF END_OF_FILE(fyle) THEN
          file_buffer := file_buffer & "//";
```

```
            put_printer(file_buffer);
            CLOSE(fyle);
            EXIT;
          ELSE
            put_printer(file_buffer);
            file_buffer := GET_LINE(fyle);
          END IF;
        END LOOP;
    END print_comment;


  PROCEDURE error(number: IN INTEGER) IS
    USE io,strlib;
    --# USE urtest;             -- for test_file
        -- processes error messages

    index: INTEGER;
    message: STRING(60);

    BEGIN
      Open(fyle,"ERRMSG",Read_Only);
      index := 1;
      LOOP
        message := Get_Line(fyle);
        IF number = index THEN
          EXIT;
        ELSIF End_of_File(fyle) THEN
          message := Int_to_Str(number) &
                     ": Not found in ERRMSG";
          EXIT;
        ELSE
          index := index + 1;
        END IF;
      END LOOP;
      Close(fyle);
      PUT(bell);
      putxy(0,22,"*** ERROR ");
      putxy(10,22,message & ". <ENTER>");
      getxy_immediate(78,22,key);
                --# remove next 2 from production
--#      PUT(test_file,"ERROR " & message);
--#      New_Line(test_file);
      putxy(0,22,blankln);
  END error;
```

```
  PROCEDURE help(data_set: IN STRING) IS
    USE io;         --Open,CLOSE,GET_LINE
--#    USE urtest;          -- test_file
        -- provides help information for a particular data
        -- item

    BEGIN
      border;
      putxy(0,3,"HELP INFORMATION - " & data_set);
      NEW_LINE;
      Open(fyle,"HELP.TXT",Read_Only);
      LOOP
        file_buffer := GET_LINE(fyle);
        IF file_buffer = data_set THEN
          LOOP
            file_buffer := GET_LINE(fyle);
            EXIT WHEN file_buffer = ".";            -- delimeter
            PUT(file_buffer); NEW_LINE; -- display line
          END LOOP;
          putxy(25,20,"<ENTER> to Continue");
          getxy_immediate(44,20,key);
          EXIT;

        ELSE
                --keep searching for data set until end
          EXIT WHEN END_OF_FILE(fyle);
        END IF;
      END LOOP;
      CLOSE(fyle);
                --# keep track of help accesses
                --# remove from production
--#      PUT(test_file,"HELP - " & data_set);
--#      NEW_LINE(test_file);
    END help;

  PROCEDURE view_table(filename: IN STRING) IS
    USE io;         -- Open,CLOSE,GET_LINE
    BEGIN
      PUT(bell);
      putxy(0,22,"<ENTER> to Continue, ENTER <X>" &
            " to View Valid Entries:");
      IF mark(52,22) THEN
        Open(fyle,filename,Read_Only);
        WHILE NOT END_OF_FILE(fyle) LOOP
          border;
          putxy(9,1,"VALID ENTRIES (REF. OPNAVINST " &
                "3503.5 " & filename &")");
          FOR y IN 3..20 LOOP
            EXIT WHEN END_OF_FILE(fyle);
            buffer := GET_LINE(fyle);
            putxy(0,y,buffer);
```

```
        END LOOP;
        putxy(0,22,"<ENTER> to Continue, ENTER <X> to " &
              "Exit:");
        EXIT WHEN mark(40,22);
      END LOOP;
      CLOSE(fyle);
    END IF;
  END view_table;



FUNCTION required_print(changed_item: BOOLEAN)
                                         RETURN BOOLEAN IS
  BEGIN
    IF action_is(draft) OR action_is(verify) OR
       (action_is(print) AND changed_item) THEN
      RETURN TRUE;
    ELSE
      RETURN FALSE;
    END IF;
  END required_print;



PROCEDURE draft_aid(data_set: IN STRING) IS
  USE printio;          --put_printer
  USE io;       --Open,CLOSE,GET_LINE,END_OF_FILE
      -- provides printed formats for the assistance
      -- of drafting a unitrep

  BEGIN
    put_printer(null_string);
    Open(fyle,"DRAFT.TXT",Read_Only);
    LOOP
      file_buffer := GET_LINE(fyle);
      IF file_buffer = data_set THEN
        put_printer("****** " & file_buffer &
                    " - DATA SET INFORMATION ******");
        LOOP
          file_buffer := GET_LINE(fyle);
          EXIT WHEN file_buffer = ".";           -- delimeter
          put_printer(file_buffer); -- printline
        END LOOP;
        put_printer(null_string);
        put_printer(null_string);
        put_printer(null_string);
        EXIT;

      ELSE
              --keep searching for data set until end
        EXIT WHEN END_OF_FILE(fyle);
      END IF;
```

```
      END LOOP;
      CLOSE(fyle);
    END draft_aid;

BEGIN
                -- strings of 1
  a_mark := "A"; x_mark := "X"; n_mark := "N";
  y_mark := "Y"; d_mark := "D"; r_mark := "R";
  blank := " "; dash := "-"; star := "*";

  null_string := "";    -- no characters
  blankln := "                                         " &
             "                                         ";
  dashln := "-----------------------------------------" &
            "-----------------------------------------";

END urglobal;
```

```
PACKAGE urutil IS
      -- This package provides utility routines to be used
      -- in constructing a UNITREP

  PROCEDURE verify_number(number_of_digits: IN INTEGER;
                          temp: IN OUT STRING;
                          success: OUT BOOLEAN;
                          number: OUT INTEGER);
      -- This procedure converts a string of numerical
      -- characters to an integer or returns an error if
      -- any character in the string is not '0'-'9'.

  PROCEDURE verify_dtg(dtg: IN OUT STRING;
                       success: OUT BOOLEAN);
      -- Error checks date-time groups.

  PROCEDURE verify_lat_long(lat_long: IN OUT STRING;
                            success: OUT BOOLEAN);
      -- Error checks lat/long position reports

  FUNCTION checksum(temp: STRING) RETURN INTEGER;
      -- performs checksums on string temp

  FUNCTION zero_pad(number_of_digits: INTEGER;
                    temp: STRING) RETURN STRING;
      -- pads a string which represents a number
      -- with zeros to fill out the field of digits

  FUNCTION getxy_digits(x,y,number_of_digits: INTEGER)
                                               RETURN STRING;
      -- gets number_of_digits at x,y and error checks

END urutil;
```

```
  WITH strlib,          -- JANUS/ADA Libraries
       consio,
       urglobal;
PACKAGE BODY urutil IS
  USE consio;
  USE urglobal;
        -- This package provides utility routines to be used
        -- in constructing a UNITREP

  PROCEDURE verify_number(number_of_digits: IN INTEGER;
                          temp: IN OUT STRING;
                          success: OUT BOOLEAN;
                          number: OUT INTEGER) IS
    USE strlib; -- Length,Str_to_Int
        -- This procedure converts a string of numerical
        -- characters to an integer or returns an error if
        -- any character in the string is not '0'-'9'.
        -- Largest valid entry is 32767.
   BEGIN
     success := TRUE;
     IF Length(temp)= 0 THEN  -- check for no input
       success := FALSE;
     ELSE           -- check for ascii '0' -'9'
       temp := zero_pad(number_of_digits,temp);
       FOR i IN 1..number_of_digits LOOP
         CASE temp(i) IS
           WHEN ' ' => temp(i) := '0';
           WHEN '0'..'9' => NULL;
           WHEN OTHERS => success := FALSE;
         END CASE;
       END LOOP;
     END IF;
     IF success THEN
       number := Str_to_Int(temp);
                -- range check
       IF number NOT IN 0..(10**number_of_digits) - 1 THEN
         success := FALSE;
       END IF;
     ELSE
       number := 0;
     END IF;
   END verify_number;
```

```
FUNCTION checksum(temp: STRING) RETURN INTEGER IS
  USE strlib;         --Extract,Str_to_Int,Length
  BEGIN
    number := 0;
    FOR i IN 1..Length(temp) LOOP
      number := number +
              Str_to_Int(Extract(temp,i,1));
      IF number >= 10 THEN
         number := number - 10;
      END IF;
    END LOOP;
    RETURN number;
  END checksum;

PROCEDURE verify_dtg(dtg: IN OUT STRING;
                     success: OUT BOOLEAN) IS
  USE strlib; --Length,Extract
      -- Error checks date-time groups.
   suc: BOOLEAN;
   temp: STRING;
   mon: STRING(3);
   dnum,hnum,mnum,ynum: INTEGER;

 BEGIN
   success := TRUE;

      -- check for incomplete string
   IF (Length(dtg) /= 12) THEN
     success := FALSE;
   ELSE

      -- rangecheck date 1 - 31
     temp := Extract(dtg,1,2);
     verify_number(2,temp,suc,dnum);
     IF ( NOT suc OR (dnum < 1) OR (dnum > 31))
      THEN success := FALSE; END IF;

      -- rangecheck hour 0 - 23
     temp := Extract(dtg,3,2);
     verify_number(2,temp,suc,hnum);
     IF (NOT suc OR (hnum < 0) OR (hnum > 23))
        THEN success := FALSE; END IF;

      -- rangecheck minute 0 - 59
     temp := Extract(dtg,5,2);
     verify_number(2,temp,suc,mnum);
     IF (NOT suc OR (mnum < 0) OR (mnum > 59))
        THEN success := FALSE; END IF;

      -- check for "Z" seperator
     temp := Extract(dtg,7,1);
```

```
     IF (temp /= "Z") THEN dtg(7) := 'Z'; END IF;

      -- rangecheck month
     mon := Extract(dtg,8,3);
     IF NOT ( (mon="JAN") OR (mon="FEB") OR (mon="MAR")
           OR (mon="APR") OR (mon="MAY") OR (mon="JUN")
           OR (mon="JUL") OR (mon="AUG") OR (mon="SEP")
           OR (mon="OCT") OR (mon="NOV") OR (mon="DEC") )
       THEN success := FALSE; END IF;

      -- rangecheck year 82 - 99
     temp := Extract(dtg,11,2);
     verify_number(2,temp,suc,ynum);
     IF ( NOT suc OR (ynum < 82) OR (ynum > 99))
       THEN success := FALSE; END IF;
   END IF;

   IF NOT success
     THEN NULL;-- range error - fall through
               -- no other checks required
      --check for combinational errors
     ELSIF ( (mon = "FEB") AND
            ( (dnum > 29) OR
              ( (dnum = 29) AND (ynum mod 4 /= 0) ) ) )
         THEN success := FALSE;
     ELSIF (((mon="NOV") OR (mon="APR") OR
             (mon="JUN") OR (mon="SEP")) AND (dnum > 30) )
         THEN success := FALSE;
     END IF;

   IF NOT success THEN
     error(1);
     dtg := "ddhhmmZMMMyy";
   END IF;
 END verify_dtg;


PROCEDURE verify_lat_long(lat_long: IN OUT STRING;
                          success: OUT BOOLEAN) IS
  USE strlib; --Length,Extract,Str_to_Int,Char_to_Str
      -- error checks lat/long positions


  suc: BOOLEAN;
  temp: STRING;
  lat_deg,lat_min,long_deg,long_min,
        lat_check,long_check: INTEGER;
  BEGIN
    success := TRUE;
```

```
IF Length(lat_long) /= 14 THEN
  success := FALSE;
ELSE
  -- check latitude
  temp := Extract(lat_long,1,2);
  verify_number(2,temp,suc,lat_deg);
  IF ( (NOT suc) OR (lat_deg NOT IN  0..90)) THEN
    success := FALSE; END IF;
  temp := Extract(lat_long,3,2);
  verify_number(2,temp,suc,lat_min);
  IF ( (NOT suc) OR (lat_min NOT IN  0..59) ) THEN
    success := FALSE; END IF;
  IF lat_long(5) /= 'N' AND lat_long(5) /= 'S' THEN
    success := FALSE; END IF;

  lat_long(7) := '-';

          -- check longitude
  temp := Extract(lat_long,8,3);
  verify_number(2,temp,suc,long_deg);
  IF ( (NOT suc) OR (long_deg NOT IN  0..180) ) THEN
    success := FALSE;
  END IF;
  temp := Extract(lat_long,11,2);
  verify_number(2,temp,suc,long_min);
  IF ( (NOT suc) OR (long_min NOT IN  0..59) ) THEN
    success := FALSE;
  END IF;
  IF lat_long(13) /= 'E' AND lat_long(13) /= 'W' THEN
    success := FALSE;
  END IF;
END IF;

IF success THEN
          -- checks degrees too large
  IF ( (Str_to_Int(Extract(lat_long,1,4)) > 90_00) OR
    (Str_to_Int(Extract(lat_long,8,5)) > 180_00) ) THEN
    success := FALSE;
  END IF;

          -- latitude checksum
  temp := Extract(lat_long,6,1);
  verify_number(1,temp,suc,lat_check);
  IF (NOT suc) OR ELSE
     lat_check /= checksum(Extract(lat_long,1,4)) THEN
      success := FALSE;
  END IF;
          -- longitude checksum
  temp := Extract(lat_long,14,1);
  verify_number(1,temp,suc,long_check);
  IF (NOT suc) OR ELSE
```

```
        long_check /= checksum(Extract(lat_long,8,5)) THEN
          success := FALSE;
      END IF;
    END IF;

    IF NOT success THEN
      error(18);      -- invalid lat_long position
      lat_long := "ddmmDC-dddmmDC";
    END IF;
  END verify_lat_long;

FUNCTION zero_pad(number_of_digits: INTEGER;
                  temp: STRING) RETURN STRING IS
  USE strlib;           -- Length
  BEGIN
    buffer := temp;
    WHILE Length(buffer) < number_of_digits LOOP
      buffer := "0" & buffer;
    END LOOP;
    RETURN buffer;
  END zero_pad;


FUNCTION getxy_digits(x,y,number_of_digits: INTEGER)
                                           RETURN STRING IS
  USE strlib;           --Char_to_Str,Extract,Length
  key: CHARACTER;
  buffer: STRING;
  success: BOOLEAN;
  number: INTEGER;
  BEGIN
    LOOP
      getxy_immediate(x,y,key);
      IF key = no_input THEN
        buffer := null_string;
        EXIT;
      ELSE
        buffer := Char_to_Str(key);
        FOR i IN 1..number_of_digits-1 LOOP
          getxy_immediate(x+i,y,key);
          EXIT WHEN key = no_input;
          buffer := buffer & Char_to_Str(key);
        END LOOP;
              -- strip trailing blanks
        FOR i IN REVERSE 1..Length(buffer) LOOP
          EXIT WHEN buffer(i) /= ' ';
          IF buffer(i) = ' ' THEN
            buffer := Extract(buffer,1,i-1);
          END IF;
        END LOOP;
        IF buffer /= null_string THEN
```

```
            verify_number(number_of_digits,
                          buffer,success,number);
            EXIT WHEN success;
            error(19);  -- invalid number
          ELSE
            EXIT;       -- return null_string
          END IF;
        END IF;
      END LOOP;
      RETURN buffer;
    END getxy_digits;

END mrutil;
```

```
PACKAGE urtest IS
        -- Provides for gathering testing data of the ROS
        -- UNITREP system

  test_file: FILE;

  PROCEDURE open_initial_test(serial: IN STRING);

  PROCEDURE open_test;  -- intermediate test

  PROCEDURE close_test; -- intermediate test

  PROCEDURE close_final_test;

END urtest;
```

```
  WITH io,strlib,                 -- JANUS/ADA libraries
       consio;
PACKAGE BODY urtest IS
  USE consio;
        -- Provides for gathering testing data of the ROS
        -- UNITREP system. Test file TESTXXX results on B:
        -- disk

test_number: INTEGER;
testee_name: STRING(30);
start_time,stop_time: STRING(12);
comment: STRING;

  PROCEDURE open_initial_test(serial: IN STRING) IS
    USE io;             -- Open,CLOSE,DELETE,Create
    USE strlib;         -- Int_to_Str
    filename: STRING(14);
    BEGIN
        -- get the test number
      Open(test_file,"TEST000",Read_Only);
      GET(test_file,test_number);
      CLOSE(test_file);
        -- increment the test number & store
      test_number := test_number + 1;
      DELETE("TEST000");
      Create(test_file,"TEST000",Write_Only);
      Open(test_file,"TEST000",Write_Only);
      PUT(test_file,test_number);
      CLOSE(test_file);
        -- open the file for THIS TEST data
      filename := "B:TEST" & Int_to_Str(test_number);
      DELETE(filename);
      Create(test_file,filename,Write_Only);
        --# should check disk full & full directory
      Open(test_file,filename.Write_Only);
        -- start test mask
      clear_screen;
      putxy(0,3,"ROS UNITREP TEST INFORMATION");
      putxy(0,6,
"ENTER Drafter Rank/Name: ------------------------------");
      putxy(0,9,
       "ENTER Start Time (Local - ex. 19DEC82 1630)");
      putxy(25,11,"ddMMMyy hhmm");
        -- get screen data
      getxy(25,6,testee_name,30);
      getxy(25,11,start_time,12);
      PUT(test_file,filename
                 &" test data on UNITREP serial " & serial);
          NEW_LINE(test_file);
      PUT(test_file,"Drafter: "& testee_name);
          NEW_LINE(test_file);
```

```
      PUT(test_file,"Start Time: "& start_time);
          NEW_LINE(test_file); NEW_LINE(test_file);
      PUT(test_file,"--------- ERRORS RAISED ----------");
          NEW_LINE(test_file);
        -- errors will be supplied from PROCEDURE error in
        -- package urglobal
    END open_initial_test;

  PROCEDURE open_test IS
    USE io;             -- Open,CLOSE,DELETE,Create
    USE strlib;         -- Int_to_Str
        -- intermediate test data
    filename: STRING(14);
    BEGIN
        -- get the test number
      Open(test_file,"TEST000",Read_Only);
      GET(test_file,test_number);
      CLOSE(test_file);
        -- increment the test number & store
      test_number := test_number + 1;
      DELETE("TEST000");
      Create(test_file,"TEST000",Write_Only);
      Open(test_file,"TEST000",Write_Only);
      PUT(test_file,test_number);
      CLOSE(test_file);
        -- open the file for THIS TEST data
      filename := "B:TEST" & Int_to_Str(test_number);
      DELETE(filename);
      Create(test_file,filename,write_Only);
        -- should check disk full & full directory
      Open(test_file,filename,Write_Only);
    END open_test;

  PROCEDURE close_test IS
    USE io;     -- CLOSE;
        -- intermediate test data
    BEGIN
      CLOSE(test_file);
    END close_test;


  PROCEDURE close_final_test IS
    USE io;     -- CLOSE
    BEGIN
        -- set close test screen
      clear_screen;
      putxy(0,3,"RCS UNITREP TEST INFORMATION");
      putxy(0,6,
        "ENTER Stop Time (Local - ex. 19DEC82 1930)");
      putxy(25,8,"ddMMMyy hhmm");
      putxy(30,10,"COMMENTS  (8 lines max)");
```

```
      getxy(25,8,stop_time,12);
      PUT(test_file,"Stop Time: "& stop_time);
      NEW_LINE(test_file);
      PUT(test_file,"*****  COMMENTS *****");
      NEW_LINE(test_file);
        -- test comment mask
      FOR y IN 11..19 LOOP
        FOR x IN 0..79 LOOP
          putxy(x,y,"-");
        END LOOP;
      END LOOP;
      FOR y IN 11..19 LOOP
        getxy(0,y,comment,80);
        EXIT WHEN comment = "";
        PUT(test_file,comment); NEW_LINE(test_file);
      END LOOP;
      CLOSE(test_file);
    END close_final_test;

END urtest;
```

```
PACKAGE urglbl IS
        -- contains structure of the comment_set used in
        -- preparation of AMPN/PMKS

        -- max number of comment lines for ampn set
  max_comment: CONSTANT := 10;

        -- for message line 69 - (AMPN/ or PMKS/) & // = 62
  max_line_length: CONSTANT := 62;

  TYPE comment_set IS
    RECORD
      change: BOOLEAN;
      number_of_lines: INTEGER; -- constrain 1..max_comment
    END RECORD;

END urglbl;
```

```
  WITH urlocalA;
PACKAGE unitrepA IS
  USE urlocalA;

          -- The basic unitrep data set and variable
          -- for data structure A, LOCAL information


  TYPE unitrep_set IS
    RECORD
      l: local_set;
    END RECORD;

    u: unitrep_set;

END unitrepA;
```

```
  WITH urgltl;          -- for comment_set
PACKAGE urlocalA IS
  USE urgltl;
        -- This package provides the data set definition
        -- for the LOCAL data used in UNITREP construction

        -- general local information
    TYPE status_set IS
      RECORD
        transmitted: BOOLEAN;
        last_serial: STRING(3);
        feeder_report: BOOLEAN;
        unit_id: STRING(30);
        current_serial: STRING(3);
        ampn: comment_set; -- for MSGID/UNITID sets
      END RECORD;

    TYPE class IS (unclas,confidential,secret,top_secret);
    TYPE prec IS (routine,priority,immediate,flash);
    TYPE message_set IS
      RECORD
        precedence: prec;
        dtg: STRING(12);
        originator_address: STRING(50);
        classification: class;
        declassification: STRING(61);
      END RECORD;

    TYPE operation_set IS
      RECORD
        underway: BOOLEAN;
        codeword: STRING(32);
        plan_org_number: STRING(23);
      END RECORD;

    TYPE exercise_set IS
      RECORD
        underway: BOOLEAN;
        nickname: STRING(62);
      END RECORD;

    TYPE position_set IS
      RECORD
        change: BOOLEAN;
        lat_long: BOOLEAN;
        present_location: STRING(20);
        dtg: STRING(12);
        ampn: comment_set;
      END RECORD;
```

```
      --maximum # of remarks pages, 10 lines/page
    max_rmks_pages: CONSTANT := 10;
    TYPE remarks_set IS
      RECORD
        change: BOOLEAN;
        number_of_pages: INTEGER;
                                  -- constrain 1..max_rmks_pages
        page: ARRAY(1..max_rmks_pages) OF comment_set;
      END RECORD;


    TYPE local_set IS
      RECORD
        change: BOOLEAN;
        status: status_set;
        message: message_set;
        operation: operation_set;
        exercise: exercise_set;
        position: position_set;
        remarks: remarks_set;
      END RECORD;

END urlocalA;
```

```
  WITH uradminB,urairB;
PACKAGE unitrepB IS
  USE uradminB,urairB;

          -- The basic unitrep data set and variable
          -- for data structure B - AIR and ADMIN


  TYPE unitrep_set IS
    RECORD
      ad: admin_set;
      ar: air_set;
    END RECORD;

    u: unitrep_set;

END unitrepB;
```

```
  WITH urglbl;          -- for comment_set
PACKAGE uradminB IS
  USE urglbl;

      -- This package provides ADMINISTRATIVE data set
      -- declarations for UNITREP construction


      -- admin TYPE declarations
    TYPE phys_status IS (onboard,departed);

    TYPE n_rep_org IS
             (cno,cincpacflt,cinclantflt,cincusnaveur);

    TYPE verification IS (valid,corrected);

    TYPE command_set IS
      RECORD
        change: BOOLEAN;
        cooic: STRING(30);
        dtg: STRING(12);
        ampn: comment_set;
      END RECORD;

    TYPE activ_set IS
      RECORD
        change: BOOLEAN;
        activity_code: STRING(2);
        ampn: comment_set;
      END RECORD;

    TYPE medic_set IS
      RECORD
        change: BOOLEAN;
        status: phys_status;
        ampn: comment_set;
      END RECORD;

    TYPE rep_org_set IS
      RECORD
        change: BOOLEAN;
        new_rep_org: n_rep_org;
        dtg: STRING(12);
        ampn: comment_set;
      END RECORD;
```

```
     TYPE verify_set IS
       RECORD
         change: BOOLEAN;
         feedback_ver: verification;
         ampn: comment_set;
        END RECORD;

     TYPE admin_set IS
       RECORD
         change: BOOLEAN;
         command: command_set;
         activ: activ_set;
         medic: medic_set;
         reporg: rep_org_set;
         verify: verify_set;
       END RECORD;

END uradmin3;
```

```
  WITH urglbl;           -- for comment_set
PACKAGE urairB IS
  USE urglbl;

        -- data structure definition for AIR Situation
        -- of a UNITREP

  max_recon: CONSTANT := 11; -- set by OPNAVINST 3503.5
  TYPE recon_set IS
    RECORD
      change: BOOLEAN;
      delete: BOOLEAN;
      number_of_capabilities: INTEGER;
      capability: ARRAY(1..max_recon) OF STRING(8);
      ampn: comment_set;
    END RECORD;

  TYPE crewstat_set IS
    RECORD
      change: BOOLEAN;
      delete: BOOLEAN;
      formed: STRING(2);
      ready: STRING(2);
      ampn: comment_set;
    END RECORD;

  TYPE airstat_set IS
    RECORD
      change: BOOLEAN;
      delete: BOOLEAN;
      possessed: STRING(2);
      fmc: STRING(2);
      pmc: STRING(2);
      nmc: string(2);
      ampn: comment_set;
    END RECORD;

  TYPE dispersed_aircraft IS
    RECORD
      change: BOOLEAN;
      delete: BOOLEAN;
      iss: STRING(30);
      airstat: airstat_set;
      crewstat: crewstat_set;
      recon: recon_set;
    END RECORD;
```

```
  TYPE authorization_set IS
    RECORD
      change: BOOLEAN;
      delete: BOOLEAN;
      iss: STRING(2);
      armpn: comment_set;
    END RECORD;

  max_locations: CONSTANT := 5; -- ROS design decision
  TYPE aircraft_type_set IS
    RECORD
      change: BOOLEAN;
      delete: BOOLEAN;
      iss: STRING(7);
      ac_auth: authorization_set;
      crews_auth: authorization_set;
      number_of_locations: INTEGER;
      location: ARRAY (1..max_locations) OF
                                         dispersed_aircraft;
    END RECORD;

  max_ac_types: CONSTANT := 5; -- ROS design decision
  TYPE air_set IS
    RECORD
      change: BOOLEAN;
      number_of_ac_types: INTEGER;
      ac_type: ARRAY (1..max_ac_types) OF aircraft_type_set;
    END RECORD;

END urair3;
```

```
  WITH io,util,chainlib,strlib, -- JANUS/ADA libraries
       consio,printio,
       filerA,
       urglobal,urutil,
--#       urtest,
       unitrepA,
       urlocalA;
PACKAGE BODY unitrep IS
  USE io,       -- Open,CLOSE,Create,DELETE
      chainlib, -- Simple_Chain
      consio,printio,
      filerA,
      urglobal,urutil,
--#      urtest,
      unitrepA,
      urlocalA;


  PROCEDURE check_required_files IS
                -- checks to make sure all neccessary files
                -- are present
    BEGIN
                -- program status
      IF NOT valid_file("STATUS") THEN
        terminate(file_error,"STATUS"); END IF;
                -- the data structure
      IF NOT valid_file("UNIT000") THEN
        terminate(file_error,"UNIT000"); END IF;
                -- serial/dtg crossref
      IF NOT valid_file("CROSSREF") THEN
        terminate(file_error,"CROSSREF"); END IF;
                -- addressees
      IF NOT valid_file("ACTION") THEN
        terminate(file_error,"ACTION"); END IF;
      IF NOT valid_file("INFO") THEN
        terminate(file_error,"INFO"); END IF;
                -- error messages
      IF NOT valid_file("ERRMSG") THEN
        terminate(file_error,"ERRMSG"); END IF;
                -- help text
      IF NOT valid_file("HELP.TXT") THEN
        terminate(file_error,"HELP"); END IF;
      IF NOT valid_file("DRAFT.TXT") THEN
        terminate(file_error,"DRAFT.TXT"); END IF;
                --# test info - remove from production
--#      IF NOT valid_file("TEST000") THEN
--#        terminate(file_error,"TEST000"); END IF;
                -- reference tables
      IF NOT valid_file("TABLE.B-2") THEN
        terminate(file_error,"TABLE.B-2"); END IF;
      IF NOT valid_file("TABLE.B-3") THEN
```

```
        terminate(file_error,"TABLE.B-3"); END IF;
      IF NOT valid_file("TABLE.B-6") THEN
       terminate(file_error,"TABLE.B-6"); END IF;
                   --module for urlocal
      IF NOT valid_file("UNITREP1.COM") THEN
        terminate(file_error,"UNITREP1.COM"); END IF;
                -- module for urair and uradmin
      IF NOT valid_file("UNITREP2.COM") THEN
        terminate(file_error,"UNITREP2.COM"); END IF;
      IF NOT valid_file("UNITREP3.COM") THEN
        terminate(file_error,"UNITREP3.COM"); END IF;
    END check_required_files;


PROCEDURE signon IS
  BEGIN
    border;
    putxy(0,3,
     "SIGN ON - ROS UNITREP SYSTEM - VERSION 1.1 - JAN 83");
    putxy(0,5,
     "The Current Workfile is UNITREP Serial --- .");
    putxy(39,5,u.l.status.current_serial);
    IF u.l.status.transmitted THEN
      putxy(0,8,"It HAS been logged as transmitted: DTG");
      putxy(39,8,u.l.message.dtg);
    ELSE
      putxy(0,8,"It HAS NOT been logged as transmitted.");
    END IF;
    putxy(10,20,
     "<ENTER> to continue, ENTER <?> for HELP information:");
    LOOP
      getxy_immediate(63,20,key);
      CASE key IS
        WHEN no_input|space => EXIT;
        WHEN ques  =>
          help("SIGN ON"); EXIT;
        WHEN OTHERS => error(6);         -- incorrect input
      END CASE;
    END LOOP;
  END signon;

  PROCEDURE choose_action(action_is: OUT action_state) IS
        -- selects one of the executable ROS UNITREP
        -- actions

    PROCEDURE initialize_action IS
      BEGIN
        FOR i IN action RANGE draft..quit LOOP
          action_is(i) := FALSE;
        END LOOP;
      END initialize_action;
```

```
PROCEDURE actionmask IS
    -- draws the action screen
  BEGIN
    border;
    putxy(68,1,"SERIAL "& u.1.status.current_serial);
    putxy(0,3,"ACTION MENU");
    putxy(19,5,"-1-  Prepare UNITREP Draft Document");
    putxy(19,6,"-2-  Initiate New UNITREP");
    putxy(19,7,
      "-3-  Remove/Modify/Add Data to Pending UNITREP");
    putxy(19,8,"-4-  Print Message Format Hard Copy " &
                "of Pending UNITREP");
    putxy(19,9,
      "-5-  Log Pending UNITREP as Transmitted");
    putxy(19,10,"-6-  Erase Pending UNITREP");
    putxy(19,11,"-7-  Prepare Verification Summary");
    putxy(19,12,"-8-  Quit");
    putxy(19,14,"ENTER the desired action: ");
    putxy(0,15,dashln);
  END actionmask;

desired_action: CHARACTER;

BEGIN
  LOOP
    initialize_action;
    actionmask;
    getxy_immediate(45,14,desired_action);
    CASE desired_action IS
      WHEN '1' => action_is(draft) := TRUE;
                  EXIT;
      WHEN '2' => action_is(neww) := TRUE;
                  EXIT;
      WHEN '3' => action_is(modify) := TRUE;
                  EXIT;
      WHEN '4' => action_is(print) := TRUE;
                  EXIT;
      WHEN '5' => action_is(log) := TRUE;
                  EXIT;
      WHEN '6' => action_is(erase) := TRUE;
                  EXIT;
      WHEN '7' => action_is(verify) := TRUE;
                  EXIT;
      WHEN '8' => action_is(quit) := TRUE;
                  EXIT;
      WHEN OTHERS => error(6);      -- incorrect input
    END CASE;
  END LOOP;
END choose_action;
```

```
PROCEDURE process_action(action_is: IN OUT action_state;
                         success: OUT BOOLEAN) IS
  USE io,strlib;
      -- performs required file actions to comply
      -- with action_is

  filename: STRING(14);
  serial: STRING(3);
  dtg: STRING(12);

  BEGIN
    program.workfile := "UNIT002";
    IF action_is(draft) THEN
      IF NOT u.l.status.transmitted THEN
        putxy(0,16,"The Current Workfile has not been " &
                   "transmitted. Do you want ...");
        putxy(5,17,"( ) Draft of the Current Workfile");
        putxy(5,18,
         "( ) Draft of the last submitted UNITREP");
        putxy(21,20,"ENTER <X> as required");
        IF mark(6,17) THEN    -- use current UNITREP data
          success := TRUE;
        ELSIF mark(6,18) THEN
              -- get last submitted UNITREP data
          filename := "B:UNIT"& u.l.status.last_serial;
          IF valid_file(filename) THEN
            program.workfile := filename;
            success := TRUE;
          ELSE                -- invalid file
            error(2);
            success := FALSE;
          END IF;
        ELSE  -- this action was not desired
          success := FALSE;
        END IF;
      ELSE                    -- use current UNITREP data
        success := TRUE;
      END IF;
      IF success THEN
        line_count := 1;      -- in printio
        set_format(text);
        putxy(0,20,blankln);
        putxy(0,20,"Select Areas for Draft Document, " &
                  "Ready printer & <ENTER>");
        getxy_immediate(59,22,key);
      END IF;

    ELSIF action_is(neww) THEN
      IF NOT u.l.status.transmitted THEN
        error(7);
              -- current not transmitted, modify or erase
```

```
    success := FALSE;
  ELSE
    success := TRUE;
  END IF;

ELSIF action_is(modify) THEN
  IF u.l.status.transmitted THEN
    error(8);     --current transmitted, initiate new
    success := FALSE;
  ELSE
    success := TRUE;
  END IF;

ELSIF action_is(print) THEN
  putxy(0,16,"Do You want a message format hard " &
             "copy of ...");
  putxy(8,17,"( ) Pending UNITREP");
  putxy(8,18,"( ) Old UNITREP: serial ---  or   " &
             "dtg: ddhhmmZMMMyy");
  putxy(17,20,"ENTER <X> and data as required.");
  IF mark(9,17) THEN      -- use current UNITREP data
    success := TRUE;
  ELSIF mark(9,18) THEN
  -- use old UNITREP data, get serial or dtg
    getxy(32,18,serial,3);
    IF serial = null_string THEN  -- get dtg
      LOOP
        getxy(46,18,dtg,12);
        verify_dtg(dtg,success);
        EXIT WHEN success;
        putxy(46,18,dtg);
      END LOOP;
      Open(fyle,"CROSSREF",Read_Only);
      LOOP
        buffer := GET_LINE(fyle);
        IF Extract(buffer,5,12) = dtg THEN
          -- dtg matches file
          CLOSE(fyle);
          filename := "B:UNIT"& Extract(buffer,1,3);
          IF valid_file(filename) THEN
            program.workfile := filename;
            success := TRUE;
            EXIT;
          ELSE            -- invalid file
            error(2);
            success := FALSE;
          END IF;
        ELSIF END_OF_FILE(fyle) THEN
          CLOSE(fyle);
          error(9); -- dtg doesn't cross to a serial
          success := FALSE;
```

```
            EXIT;
          ELSE
            NULL;   -- get another line from CROSSREF
          END IF;
        END LOOP;
      ELSE              -- use input serial
        filename := "B:UNIT" & serial;
        IF valid_file(filename) THEN
          load_file(filename);
          program.workfile := filename;
          success := TRUE;
        ELSE
          error(2);             -- invalid file
          success := FALSE;
        END IF;
      END IF;
    ELSE                        -- not the desired action
      success := FALSE;
    END IF;
                    -- pick message format
    IF success THEN
      FOR i IN 16..20 LOOP
        putxy(0,i,blankln);
      END LOOP;
      putxy(0,16,"DO you want the message in ...");
      putxy(8,17,"( ) OCR format");
      putxy(8,18,"( ) Standard Naval Message format");
      putxy(27,20,"ENTER <X>");
      LOOP
        IF mark(9,17) THEN
          set_format(ocr); EXIT;
        ELSIF mark(9,18) THEN
          set_format(msg); EXIT;
        ELSE
          error(6);             -- choose one
        END IF;
      END LOOP;
      line_count := 1;          -- in printio
      putxy(20,20,"Ready printer and <ENTER>");
      getxy_immediate(44,20,key);
    END IF;

ELSIF action_is(log) THEN
  IF u.l.status.transmitted THEN
    error(10);
    -- cannot log an already transmitted UNITREP
    success := FALSE;
  ELSE
    putxy(0,16,"ENTER the transmittal dtg: " &
               "ddhhmmZMMMyy");
    LOOP
```

```
  getxy(27,16,u.l.message.dtg,12);
  verify_dtg(u.l.message.dtg,success);
  EXIT WHEN success;
  putxy(27,16,u.l.message.dtg);
END LOOP;
u.l.status.transmitted := TRUE;
u.l.status.last_serial := u.l.status.current_serial;
filename := "B:UNIT"& u.l.status.current_serial;
program.workfile := filename;
store_file(filename);   -- completed data file
store_file("UNIT000");  -- workfile
--* should incorporate disk full,
--* full directory checks

-- the insert_dtg block is neccessary due to the
-- lack of random access read/write procedures in
-- JANUS ADA. Greatly simplified in full ADA
insert_dtg: DECLARE
  TYPE list;
  TYPE link IS ACCESS list;
  TYPE list IS
    RECORD
      serial_dtg: STRING(16);
      next: link;
    END RECORD;
  mark,first,l: link;

  BEGIN
    first := NEW list;
        -- initialize - not required in full ADA
    first.serial_dtg := u.l.status.current_serial &
                        " " & u.l.message.dtg;
    first.next := NULL;
    mark := first;
    Open(fyle,"CROSSREF",Read_Only);
                -- read out old crossref data
    WHILE NOT END_OF_FILE(fyle) LOOP
      l := NEW list;
          -- initialize - not required in full ADA
       l.serial_dtg := GET_LINE(fyle);
       l.next := null;
      mark.next := l;
      mark := l;
    END LOOP;
    CLOSE(fyle);
        -- write in the altered crossref list
    DELETE("CROSSREF");
    Create(fyle,"CROSSREF",Write_Only);
    mark := first;
    WHILE mark /= NULL LOOP
      PUT(fyle,mark.serial_dtg); New_Line(fyle);
```

```
          mark := mark.next;
        END LOOP;
        CLOSE(fyle);
                    -- Clean up pointers
        WHILE first /= NULL LOOP
          mark := first;
          first := mark.next;
          Dispose(mark);
        END LOOP;
      END insert_dtg;
      success := TRUE; -- DELETE old AMP files on a:
   END IF;

ELSIF action_is(erase) THEN
    -- if successful, erases current workfile and
    -- returns false. reselect choice for action
  IF NOT u.l.status.transmitted THEN
    putxy(10,16,"Confirm you wish to erase UNITREP " &
                "serial " & u.l.status.current_serial);
    putxy(13,17,
     "(It has not been logged as transmitted)");
    putxy(26,18,"ENTER (Y/N):");
    getxy_immediate(38,18,key);
    IF key = n THEN           -- do not erase
      success := FALSE;
    ELSIF key = y THEN
    -- erase if backup data is available
      filename := "B:UNIT" & u.l.status.last_serial;
      IF valid_file(filename) THEN
        program.workfile := filename;
        load_file(filename);
        store_file("UNIT000");
        success := TRUE;      -- load partB
      ELSE
        terminate(file_error,filename);
      END IF;
    ELSE
      error(6);       -- incorrect input
      success := FALSE;
    END IF;
  ELSE                -- file has been transmitted
    error(8);         -- initiate new
    success := FALSE;
  END IF;

ELSIF action_is(verify) THEN
  putxy(0,16,
   "ENTER the AS_OF verification serial: ---");
  putxy(9,17,"(000 to restart)");
  serial := getxy_digits(37,16,3);
  IF number = 0 THEN
```

```
      success := FALSE;       -- restart on 000
    ELSE
      filename := "B:UNIT" & serial;
      IF valid_file(filename) THEN
        success := TRUE;
        program.workfile := filename;
      ELSE
        error(2);     --required file not present
        success := FALSE;
      END IF;
    END IF;
    IF success THEN
      line_count := 1;        -- in printio
      set_format(msg);
      putxy(0,20,blankln);
      putxy(35,20,"Ready Printer and <ENTER>");
      getxy_immediate(59,20,key);
    END IF;

  ELSIF action_is(quit) THEN
    terminate(normal,null_string);

  ELSE                        -- no action required
    success := FALSE;
  END IF;

END process_action;


PROCEDURE choose_areas(area_is: IN OUT area_state) IS
      -- menu selection of the UNITREP situation
      -- area categories

  PROCEDURE initialize_area IS
    BEGIN
      FOR i IN areas RANGE pers..local LOOP
        area_is(i) := FALSE;
      END LOOP;
   END initialize_area;

  PROCEDURE areamask IS
      -- draws the area screen
    BEGIN
      border;
      putxy(68,1,"SERIAL "& u.1.status.current_serial);
      putxy(0,3,"UNITREP SITUATION MENU");
      putxy(9,5,"( ) -0-  Modify Local Information");
      putxy(9,7,"( ) -1-  Personnel Status");
      putxy(9,8,"( ) -2-  Administrative Status");
      putxy(9,9,
        "( ) -3-  Unit Combat Readiness Assessment");
```

```
        putxy(9,10,"( ) -4-  Aircraft and Crews Status");
        putxy(9,11,"( ) -5-  Major Equipment Status");
        putxy(9,12,"( ) -6-  Special Capabilities Status");
        putxy(9,13,
         "( ) -7-  Increased Defense Readiness Status");
        putxy(9,15,"( ) -8-  Reserve Augmentation Status");
        putxy(9,16,"( ) -9-  Type Commander Reports");
        putxy(15,20,"ENTER <X> in desired report area(s)");
      END areamask;

    BEGIN
      LOOP
        initialize_area;
        areamask;
        area_is(local) := mark(10.5);
        area_is(pers) := mark(10,7);
        area_is(admin) := mark(10,8);
        area_is(combat) := mark(10,9);
        area_is(air) := mark(10,10);
        area_is(equip) := mark(10,11);
        area_is(specap) := mark(10,12);
        area_is(defense) := mark(10,13);
        area_is(reserve) := mark(10,15);
        area_is(tycom) := mark(10,16);
        IF area_is(pers) OR area_is(combat) OR
           area_is(equip) OR area_is(specap) OR
           area_is(defense) OR area_is(reserve)
           OR area_is(tycom) THEN
           putxy(0,20,
           "SELECTED AREA NOT IMPLEMENTED - CHOOSE ONLY: " &
                  "LOCAL,ADMINISTRATIVE and/or AIR");
           getxy_immediate(78,20,key);
        ELSE
          putxy(0,20,blankln);
          putxy(0,20,
           "<ENTER> to continue, ENTER <X> to reselect: ");
          reselect := mark(44,20);
          EXIT WHEN NOT reselect;
        END IF;
      END LOOP;
    END choose_areas;


BEGIN  -- main body
  load_status;
                -- operations required by status
  IF program.initial_entry THEN
    check_required_files;
                        --# initial test data
--#    open_initial_test(u.l.status.current_serial);
    signon;
```

```
    program.initial_entry := FALSE;
  ELSE
--#     open_test;        --# intermediate test data
    load_file("UNIT000");
  END IF;

  action: LOOP
    choose_action(action_is);
    process_action(action_is,success);
        -- required file operations
    EXIT WHEN success;
  END LOOP action;

  IF action_is(modify) OR action_is(draft) THEN
    choose_areas(area_is);
  END IF;

        -- direct program flow to next required module
--#  close_test;
  store_status;
  putxy(30,22,"STANDBY.....");
  IF action_is(log) THEN
      -- reset local items in UNIT000A
    Simple_Chain("UNITREP1.COM");
  ELSIF action_is(erase) THEN
      -- recall/reset items in UNIT000B. UNIT000A reset
      -- in process_action
    Simple_Chain("UNITREP2.COM");
  ELSIF action_is(print) OR action_is(draft)
        OR action_is(verify) OR action_is(neww) THEN
    Simple_Chain("UNITREP1.COM");
  ELSIF action_is(modify) AND
        (NOT u.l.change OR area_is(local) ) THEN
                -- At least one entry into urlocal is
                -- required to set message items
    Simple_Chain("UNITREP1.COM");
  ELSIF action_is(modify) AND area_is(admin) THEN
    Simple_Chain("UNITREP2.COM");
  ELSIF action_is(modify) AND area_is(air) THEN
    Simple_Chain("UNITREP3.COM");
  END IF;
  Simple_Chain("UNITREP.COM"); -- restart

END unitrep;
```

```
  WITH chainlib,                  -- JANUS ADA Library
       consio,printio,
       filerA,
--#       urtest,
       urglobal,
       urlocal;
PACKAGE BODY unitrep1 IS
  USE chainlib,
      consio,printio,
      filerA,
--#      urtest,
      urglobal,
      urlocal;

        -- Provides processing of UNITREP local
        -- information and program control

  BEGIN
    load_status;
    IF program.print_trailer THEN
      process_local(trailer);
      program.print_trailer := FALSE;
      store_status;
      Simple_Chain("UNITREP.COM");
    ELSIF action_is(log) THEN
      process_local(header);
      store_file(program.workfile);
      Simple_Chain("UNITREP2.COM");
    ELSIF action_is(print) OR action_is(verify) THEN
      process_local(header);
      program.print_trailer := TRUE;
      store_status;               -- line_count
      Simple_Chain("UNITREP2.COM");
    ELSIF action_is(draft) THEN
      process_local(header);
      program.print_trailer := TRUE;
      IF area_is(admin) or area_is(air) THEN
        store_status;             -- line_count
        Simple_Chain("UNITREP2.COM");
      ELSE
        process_local(trailer);
        program.print_trailer := FALSE;
        Simple_Chain("UNITREP.COM");
      END IF;
    ELSIF action_is(neww) THEN
      process_local(header);
      store_file("UNIT000");
      Simple_Chain("UNITREP2.COM");
    ELSIF action_is(modify) THEN
--#      open_test;
      process_local(header);
```

```
--#      close_test;
      store_file("UNIT000");
      store_status;
      putxy(30,22,"STANDBY.....");
      IF area_is(admin) THEN
        Simple_Chain("UNITREP2.COM");
      ELSIF area_is(air) THEN
        Simple_Chain("UNITREP3.COM");
      ELSE
        Simple_Chain("UNITREP.COM");
      END IF;
        -- erase controlled in unitrep.com (.pkg)
    END IF;
    Simple_Chain("UNITREP.COM"); -- restart

  END unitrep1;
```

```
  WITH chainlib,                   -- JANUS/ADA library
       consio,printio,
       filer3,
       urglobal,
--#       urtest,
       unitrep3,
       uradmin,urair;
PACKAGE BODY unitrep2 IS
  USE chainlib,
      consio,printio,
      filer3,
      urglobal,
--#      urtest,
      unitrep3,
      uradmin,urair;
        -- provides processing of UNITREP administrative and
        -- air situation reports (excluding air modify) and
        -- program control

  BEGIN
    load_status;
    IF action_is(verify) OR action_is(print) THEN
      process_administrative;
      process_air;
      store_status;                -- for line_count
      Simple_Chain("UNITREP1.COM");      -- print trailer
    ELSIF action_is(draft) THEN
      IF area_is(admin) THEN
        process_administrative;
      END IF;
      IF area_is(air) THEN
        process_air;
      END IF;
      store_status;                -- for line_count
      Simple_Chain("UNITREP1.COM");      -- print_trailer
    ELSIF action_is(neww) THEN
      process_administrative;
      process_air;
      store_file("UNIT000");
      Simple_Chain("UNITREP.COM");       -- file zeroed
    ELSIF action_is(modify) THEN
--#      open_test;
      IF area_is(admin) THEN
        process_administrative;
        store_file("UNIT000");  -- save new data
      END IF;
      IF area_is(air) THEN
--#     close_test;
        putxy(35,22,"STANDBY.....");
        Simple_Chain("UNITREP3.COM");    -- process_ac_type
      END IF;
```

```
--#      close_test;            --# remove from production
     putxy(35,22,"STANDBY.....");
     Simple_Chain("UNITREP.COM");
   ELSIF action_is(log) THEN
     process_administrative;
     process_air;
     store_file(program.workfile);
     Simple_Chain("UNITREP.COM");
   ELSIF action_is(erase) THEN
     store_file("UNIT000"); -- previously loaded old file
     Simple_Chain("UNITREP.COM");
   END IF;
   Simple_Chain("UNITREP.COM"); -- restart

  END unitrep2;
```

```
  WITH chainlib,                     -- JANUS/ADA library
       consio,printio,
       urglobal,
       filerB,
--#       urtest,
       urair1,
       unitrepB;
PACKAGE BODY unitrep3 IS
  USE chainlib,
      consio,printio,
      urglobal,
      filerB,
--#      urtest,
      urair1,
      unitrepB;
        -- provides processing of UNITREP air situation
        -- reports when action_is(modify) and program
        -- control to return to main module

  BEGIN
    load_status;
--#    open_test;
    process_ac_type;
--#    close_test;
    store_file("UNIT000");
    putxy(35,22,"STANDBY.....");
    Simple_Chain("UNITREP.COM");
  END unitrep3;
```

```
PACKAGE filerA IS

        -- normal file handling operations for the 'A' data
        -- structure

  FUNCTION valid_file(filename: STRING) RETURN BOOLEAN;
        -- checks for valid external files

  TYPE reason_code IS (normal,file_error);
  PROCEDURE terminate(reason: IN reason_code;
                      filename: IN STRING);
        -- exit point for the UNITREP program

  PROCEDURE load_file(filename: IN STRING);
        -- loads a particular unitrep data set

  PROCEDURE store_file(filename: IN STRING);
        -- stores a particular unitrep data set

  PROCEDURE load_status;
        -- loads and performs operations required by status

  PROCEDURE store_status;
        -- stores the status state

END filerA;
```

```
  WITH io,util,strlib,  --JANUS/ADA library
       consio,printio,
       urglobal,
--#       urtest,
       unitrepA,
       urlocalA;
PACKAGE BODY filerA IS
  USE io;
  USE printio;            -- format_is
  USE urglobal;           --action_is,area_is,format_is
  USE unitrepA;           --u,data types
  USE urlocalA;
        -- normal file handling operations for the 'A' data
        -- structure

  fyle: FILE;

  FUNCTION valid_file(filename: STRING) RETURN BOOLEAN IS
    USE io;       -- Open,CLOSE,IOresult
    USE strlib; -- Extract
        -- opens FILENAME to check presence.If FILENAME
        -- not present, returns FALSE otherwise closes
        -- FILENAME and returns TRUE.
    BEGIN
               -- A unitrep file
      buffer := filename;
      IF ((Extract(buffer,1,4) = "UNIT") OR
         (Extract(buffer,1,4) = "B:UN"))
                AND NOT
        ((filename = "UNITREP1.COM") OR
         (filename = "UNITREP2.COM") OR
         (filename = "UNITREP3.COM"))     THEN
               -- due to check_required_files
        buffer := buffer & "A";
      END IF;
      Open(fyle,buffer,Read_Only);
      IF IOresult = 255 THEN    -- file does not exist
        RETURN FALSE;
      ELSE
        CLOSE(fyle);
        RETURN TRUE;
      END IF;
    END valid_file;

  PROCEDURE terminate(reason: IN reason_code;
                    filename: IN STRING) IS
    USE consio;
    USE strlib;           -- Extract
    USE io,util;    --HALT
--#    USE urtest;                -- test_file
        -- This procedure is the exit point of the program
```

```
        -- reason can be either NORMAL or for FILE_ERROR
    BEGIN
                -- save current unitrep data
      store_file("UNIT000");
      IF reason = normal THEN
--#        close_final_test;
        program.initial_entry := TRUE;
        store_status;
        Halt;
      ELSE      -- reason is file_error
        buffer := filename;
        IF (Extract(buffer,1,4) = "UNIT") AND NOT
           ((filename = "UNITREP1.COM") OR
            (filename = "UNITREP2.COM") )    THEN
                -- in check_required_files
          buffer := buffer & "A";
        END IF;
                -- initial file abort
--#     IF NOT IS_OPEN(test_file) THEN
--#       Open(test_file,"TEST.ABR",Write_Only);
--#     END IF;
--#     PUT(test_file,
--#      "FILE ERROR TERMINATION - " & filename);
--#     NEW_LINE(test_file); NEW_LINE(test_file);
--#     CLOSE(test_file);
                                 --
        clear_screen;
        putxy(0,3,"FILE ERROR TERMINATION");
        putxy(2,5,buffer & " is required " &
                  "but not present on the assigned disk");
        putxy(0,6,
         "Copy the required file to the proper disk.");
        Halt;
      END IF;
    END terminate;

  PROCEDURE load_file(filename: IN STRING) IS
        -- loads a particular unitrep data set
    BEGIN
      buffer := filename & "A";
      Open(fyle,buffer,Read_Only);
      Read(fyle,u);
      CLOSE(fyle);
    END load_file;

  PROCEDURE store_file(filename: IN STRING) IS
        -- stores a particular unitrep data set
    BEGIN
      buffer := filename & "A";
      DELETE(buffer);
      Create(fyle,buffer,Write_Only);
```

```
    Write(fyle,u);
    CLOSE(fyle);
  END store_file;

 PROCEDURE load_status IS
      -- loads and performs operations required by status
  BEGIN
    Open(fyle,"STATUS",Read_Only);
    Read(fyle,program);
    CLOSE(fyle);
    action_is := program.current_action;
    area_is := program.current_area;
    format_is := program.current_format;
      IF format_is(msg) THEN
        set_format(msg);
      ELSIF format_is(ocr) THEN
        set_format(ocr);
      ELSE    -- format is text
        set_format(text);
      END IF;
    line_count := program.printing_line;
    IF action_is(log) THEN
      load_file("UNIT000");
    ELSE
      load_file(program.workfile);
    END IF;
  END load_status;

 PROCEDURE store_status IS
      -- stores the status state
  BEGIN       .
      -- .unit set in urlocal, process_local_infrequent
      -- .initial_entry set in unitrep (quit) and
              -- main body
      -- .print_trailer set in unitrep main body
    program.message_transmitted := u.l.status.transmitted;
    program.current_action := action_is;
      -- set in unitrep.choose_action
    program.current_area := area_is;
      -- set in unitrep.choose_area
    program.current_format := format_is;
      -- set in printio.set_format
    program.printing_line := line_count;
      -- set in printio.put_printer
      -- .workfile set in unitrep.choose_action
    DELETE("STATUS");
    Create(fyle,"STATUS",Write_Only);
    Write(fyle,program);
    CLOSE(fyle);
  END store_status;
END filerA;
```

```
PACKAGE filerB IS

        -- normal file handling operations for the 'B' data
        -- structure

  FUNCTION valid_file(filename: STRING) RETURN BOOLEAN;
        -- checks for valid external files

  TYPE reason_code IS (normal,file_error);
  PROCEDURE terminate(reason: IN reason_code;
                      filename: IN STRING);
        -- exit point for the UNITREP program

  PROCEDURE load_file(filename: IN STRING);
        -- loads a particular unitrep data set

  PROCEDURE store_file(filename: IN STRING);
        -- stores a particular unitrep data set

  PROCEDURE load_status;
        -- loads and performs operations required by status

  PROCEDURE store_status;
        -- stores the status state

END filerB;
```

```
  WITH io,util,strlib,  --JANUS/ADA library
       consio,printio,
       urglobal,
--#       urtest,
       unitrepB,
       uradminB,urairB;
PACKAGE BODY filerB IS
  USE io;
  USE printio;          -- format_is
  USE urglobal;         -- action_is,area_is
  USE unitrepB;         --u,data types
  USE uradminB,urairB;

        -- normal file handling operations for the 'B' data
        -- structure

  fyle: FILE;


  FUNCTION valid_file(filename: STRING) RETURN BOOLEAN IS
    USE io;     -- Open,CLOSE,IOresult
    USE strlib; -- Extract
        -- opens FILENAME to check presence.If FILENAME
        -- not present, returns FALSE otherwise closes
        -- FILNAME and returns TRUE
    BEGIN
                -- B unitrep file
      buffer := filename;
                -- check for UNITxxxB files
      IF ((Extract(buffer,1,4) = "UNIT") OR
               (Extract(buffer,1,4) = "B:UN")) THEN
        buffer := buffer & "B";
      END IF;
      Open(fyle,buffer,Read_Only);
      IF IOresult = 255 THEN    -- file does not exist
        RETURN FALSE;
      ELSE
        CLOSE(fyle);
        RETURN TRUE;
      END IF;
    END valid_file;


  PROCEDURE terminate(reason: IN reason_code;
                      filename: IN STRING) IS
    USE consio;
    USE io,util;    -- Halt
--#    USE urtest;             -- test_file
        -- This procedure is the exit point of the program
        -- reason can be either NORMAL or for FILE_ERROR
    BEGIN
```

```
                -- save current unitrep data
      store_file("UNIT000");
      IF reason = normal THEN
--#        close_final_test;
        Halt;
      ELSE      -- reason is file_error
                -- check_required_files abort
--#     IF NOT IS_OPEN(test_file) THEN
--#       Open(test_file,"TEST.ABR",Write_Only);
--#     END IF;
--#     PUT(test_file,
--#      "FILE ERROR TERMINATION - " & filename);
--#     NEW_LINE(test_file); NEW_LINE(test_file);
--#     CLOSE(test_file);
                                 --
        clear_screen;
        putxy(0,3,"FILE ERROR TERMINATION");
        putxy(0,5,filename & " is required " &
                  "but not present on the assigned disk");
        putxy(0,6,
         "Copy the required file to the proper disk.");
        Halt;
      END IF;
    END terminate;

  PROCEDURE load_file(filename: IN STRING) IS
        -- loads a particular unitrep data set
    BEGIN
      buffer := filename & "B";
      Open(fyle,buffer,Read_Only);
      Read(fyle,u);
      CLOSE(fyle);
    END load_file;

  PROCEDURE store_file(filename: IN STRING) IS
        -- stores a particular unitrep data set
    BEGIN
      buffer := filename & "B";
      DELETE(buffer);
      Create(fyle,buffer,Write_Only);
      Write(fyle,u);
      CLOSE(fyle);
    END store_file;

  PROCEDURE load_status IS
        -- loads and performs operations required by status
    BEGIN
      Open(fyle,"STATUS",Read_Only);
      Read(fyle,program);
      CLOSE(fyle);
      action_is := program.current_action;
```

```
      area_is := program.current_area;
      format_is := program.current_format;
        IF format_is(msg) THEN
          set_format(msg);
        ELSIF format_is(ocr) THEN
          set_format(ocr);
        ELSE    -- format is text
          set_format(text);
        END IF;
      line_count := program.printing_line;
      IF NOT valid_file("UNIT000") THEN
        terminate(file_error,"UNIT000"); -- check UNIT000B
      END IF;
      IF action_is(log) THEN
        load_file("UNIT000");
      ELSE
        load_file(program.workfile);
      END IF;
    END load_status;

  PROCEDURE store_status IS
        -- stores the status state
    BEGIN
        -- .unit set in urlocal, process_local_infrequent
        -- .initial_entry set in unitrep (quit)
            -- and main body
        -- .print_trailer set in unitrep main body
        -- .message_transmitted set by filerA
      program.current_action := action_is;
        -- set in unitrep.choose_action
      program.current_area := area_is;
        -- set in unitrep.choose_area
      program.current_format := format_is;
        -- set in printio.set_format
      program.printing_line := line_count;
        -- set in printio.put_printer
        -- .workfile set in unitrep.choose_action
      DELETE("STATUS");
      Create(fyle,"STATUS",Write_Only);
      Write(fyle,program);
      CLOSE(fyle);
    END store_status;

END filerB;
```

```
  WITH unitrepA,urlocalA;
PACKAGE urlocal IS
  USE unitrepA,urlocalA;

        -- This package provides LOCAL routines for
        -- UNITREP construction

  TYPE msg_part IS (header,trailer);
  PROCEDURE process_local(part: IN msg_part);

END urlocal;
```

```
  WITH io,strlib,          --JANUS/ADA libraries
      consio,printio,
      urutil,
      urgltl,urglobal,
      urlocalA,urlocal1,
      unitrepA;
PACKAGE BODY urlocal IS
  USE consio,printio,
      urutil,
      urgltl,urglobal,
      urlocalA,urlocal1,
      unitrepA;

        -- This package provides LOCAL routines for
        -- UNITREP construction
        -- Data sets MSGID/UNITID, OPER, EXER, POSIT,
        -- RMKS are supported here.
        -- Message information and DCLAS supported in
        -- urlocal1

  PROCEDURE process_local(part: IN msg_part) IS
    USE strlib;            --Str_to_Int,Int_to_Str
    USE io;                --Open,CLOSE,GET_LINE


    PROCEDURE process_local_misc IS
      USE strlib;                 -- Extract

      PROCEDURE localmask_misc IS
        BEGIN
          localmask;
          putxy(30,3,"-MISCELLANEOUS-");
          putxy(0,4,"( )      -1a- Operation Underway");
          putxy(1,5,"|"); putxy(12,5,"Operation Codeword");
          putxy(47,5,"Plan Originator and Number");
          putxy(1,6,"or");
          putxy(6,6,"--------------------------------");
          putxy(48,6,"------------------------");
          putxy(1,7,"|");
          putxy(0,8,"( )      -1b- Exercise Underway");
          putxy(1,9,"|"); putxy(30,9,"Exercise Nickname");
          putxy(1,10,"or");
          putxy(8,10,"-------------------------------"&
                     "-------------------------------");
          putxy(1,11,"|");
          putxy(0,12,
            "( )      -1c- Operation/Exercise Terminated");
          putxy(0,14,
            "( )      -2- Position - Required for ship/sub");
          putxy(10,15,
            "Location (or EXEMPT)  or    LAT/LONG");
```

```
    putxy(57,15,"As Of DTG");
    putxy(10,16,
       "-------------------     ddmmDC-dddmmDC");
    putxy(55,16,"ddhhmmZMMMyy");
    putxy(0,18,"( )     -3- Remarks:");
  END localmask_misc;

PROCEDURE fill_local_misc IS
  BEGIN
    localmask_misc;
    IF u.l.operation.underway THEN
      putxy(4,4,star);
      putxy(6,6,u.l.operation.codeword);
      putxy(48,6,u.l.operation.plan_org_number);
    ELSIF u.l.exercise.underway THEN
      putxy(4,8,star);
      putxy(8,10,u.l.exercise.nickname);
    ELSE
      NULL;
    END IF;

    IF NOT u.l.position.lat_long THEN
      putxy(10,16,u.l.position.present_location);
    ELSE
      putxy(33,16,u.l.position.present_location);
    END IF;
    putxy(55,16,u.l.position.dtg);
    IF u.l.position.change THEN
      putxy(4,14,star);
      IF u.l.position.ampn.change THEN
        putxy(6,14,a_mark);
      END IF;
    END IF;

    IF u.l.remarks.change THEN
      putxy(4,18,star);
      putxy(21,18,"ON THIS PAGE");
    ELSE
      putxy(21,18,"NONE");
    END IF;
  END fill_local_misc;

PROCEDURE process_remarks IS
          -- processes remarks sets


  BEGIN
    u.l.remarks.number_of_pages := 0;
    FOR number IN 1..max_rmks_pages LOOP
      process_comment("RMKS" & Int_to_Str(number),
                          u.l.remarks.page(number));
```

```
      IF NOT u.l.remarks.page(number).change THEN
        EXIT;     -- last page not modified
      ELSIF u.l.remarks.page(number).number_of_lines <
                                      max_comment THEN
        u.l.remarks.number_of_pages := number;
        EXIT;     -- last page not completely used
      ELSE
        u.l.remarks.number_of_pages := number;
      END IF;
    END LOOP;
    IF u.l.remarks.number_of_pages > 0 THEN
      u.l.remarks.change := TRUE;
    ELSE
      u.l.remarks.change := FALSE;
    END IF;
  END process_remarks;

BEGIN
  LOOP
    fill_local_misc;
                  --operation/exercise
    modify_oper_exer: LOOP
      IF NOT u.l.exercise.underway THEN
        LOOP
          putxy(1,4,blank);
          getxy_immediate(1,4,key);
          CASE key IS
            WHEN x|tab =>
              clear_field(6,6,32);
              clear_field(48,6,23);
              getxy(6,6,buffer,32);
              IF buffer = null_string THEN
                u.l.operation.underway := FALSE;
              ELSE
                u.l.operation.codeword := buffer;
                u.l.operation.underway := TRUE;
                putxy(4,4,star);
                getxy(48,6,
                    u.l.operation.plan_org_number,23);
                EXIT modify_oper_exer;
              END IF;
            WHEN a =>
              error(29);  -- ampn not allowed
            WHEN no_input|space =>
              EXIT;
            WHEN ques =>
              help("OPER/EXER");
              fill_local_misc;
            WHEN OTHERS =>
              error(6);   --incorrect input
          END CASE;
```

```
  END LOOP;
END IF;

IF NOT u.l.operation.underway THEN
  LOOP
    putxy(1,8,blank);
    getxy_immediate(1,8,key);
    CASE key IS
      WHEN x|tab =>
        clear_field(8,10,62);
        getxy(8,10,buffer,62);
        IF buffer = null_string THEN
          u.l.exercise.underway := FALSE;
          EXIT;
        ELSE
          u.l.exercise.nickname := buffer;
          u.l.exercise.underway := TRUE;
          putxy(4,8,star);
          EXIT modify_oper_exer;
        END IF;
      WHEN a =>
        error(29);  -- ampn not allowed
      WHEN no_input|space =>
        EXIT;
      WHEN ques =>
        help("OPER/EXER");
        fill_local_misc;
      WHEN OTHERS =>
        error(6);           -- incorrect input
    END CASE;
  END LOOP;
END IF;

LOOP
  putxy(1,12,blank);
  getxy_immediate(1,12,key);
  CASE key IS
    WHEN x|tab =>
      u.l.operation.underway := FALSE;
      putxy(4,4,blank);
      clear_field(6,6,32);
      clear_field(48,6,23);
      u.l.exercise.underway := FALSE;
      putxy(4,8,blank);
      clear_field(8,10,62);
      EXIT modify_oper_exer;
    WHEN a =>
      error(29);    -- ampn not allowed
    WHEN no_input|space =>
      EXIT modify_oper_exer;
    WHEN ques =>
```

```
        help("OPER/EXER");
        fill_local_misc;
      WHEN OTHERS =>
        error(6);     -- incorrect input
    END CASE;
  END LOOP;

END LOOP modify_oper_exer;

modify_position: LOOP
  putxy(1,14,blank);
      -- position is required for ship/sub, only
      -- when changed for others
  IF ((program.unit IN ship..submarine)
              AND NOT (u.l.position.change)) THEN
    key := x;
  ELSE
    getxy_immediate(1,14,key);
  END IF;
  CASE key IS
    WHEN x|tab =>
      putxy(4,14.star);
      clear_field(10,16,20);
      putxy(33,16,"ddmmIC-dddmmDC");
      putxy(55,16,"ddhhmmZMMMyy");
      getxy(10,16,
              u.l.position.present_location,20);
      u.l.position.change := TRUE;
      IF u.l.position.present_location = "EXEMPT"
                                              THEN
        NULL; -- no further action required
      ELSIF u.l.position.present_location =
                                  null_string THEN
              -- get lat/long
        u.l.position.lat_long := TRUE;
        LOOP
          getxy(35,16,
                u.l.position.present_location,14);
          verify_lat_long(
           u.l.position.present_location,success);
          EXIT WHEN success;
          putxy(35,16,
                   u.l.position.present_location);
        END LOOP;
                      -- get asof dtg
        LOOP
          getxy(55,16,u.l.position.dtg,12);
          verify_dtg(u.l.position.dtg,success);
          EXIT WHEN success;
          putxy(55,16,u.l.position.dtg);
        END LOOP;
```

```
            ELSE
              u.l.position.lat_long := FALSE;
                            -- get asof dtg
              LOOP
                getxy(55,16,u.l.position.dtg,12);
                verify_dtg(u.l.position.dtg,success);
                EXIT WHEN success;
                putxy(55,16,u.l.position.dtg);
              END LOOP;
            END IF;
          WHEN a =>
            IF u.l.position.change THEN
              process_comment("POSIT",
                                     u.l.position.ampn);
              fill_local_misc;
            ELSE
              error(27);    -- set not changed
            END IF;
          WHEN no_input|space =>
            EXIT modify_position;
          WHEN ques =>
            help("POSIT");
            fill_local_misc;
          WHEN OTHERS =>
            error(28);      +-- allowable inputs
        END CASE;
      END LOOP modify_position;

                    -- remarks
      IF mark(1,18) THEN
        process_remarks;
        fill_local_misc;
      END IF;

      putxy(0,20,blankln);
      putxy(0,20,"<ENTER> to continue, ENTER <X> " &
                 "to Reselect: ");
      reselect := mark(44,20);
      EXIT WHEN NOT reselect;
    END LOOP;
  END process_local_misc;


PROCEDURE process_local_infrequent IS

  PROCEDURE localmask_infrequent IS
    BEGIN
      localmask;
      putxy(24,3,"-INFREQUENTLY CHANGED ITEMS-");
      putxy(0,5,"( ) -1- Unit Type");
```

```
    putxy(8,6,"( )Ship   ( )Submarine   ( )Air   " &
              "( )Shore   ( )Other");
    putxy(0,9,"( ) -2- Unit Identification - " &
               "OPNAVINST 3503.1 APP. C");
    putxy(8,10,"------------------------------");
    putxy(0,12,"( ) -3- Last Serial ---");
    putxy(0,15,"( )   -4- Message to be Prepared " &
                 "as Feeder:");
    putxy(12,16,"(UNITID vice MSGID data set)");
  END localmask_infrequent;

PROCEDURE fill_local_infrequent IS
  BEGIN
    localmask_infrequent;
    CASE program.unit IS
      WHEN ship => putxy(9,6,x_mark);
      WHEN submarine => putxy(19,6,x_mark);
      WHEN air_unit => putxy(34,6,x_mark);
      WHEN shore => putxy(43,6,x_mark);
      WHEN other => putxy(54,6,x_mark);
    END CASE;
    putxy(8,10,u.l.status.unit_id);
    putxy(20,12,u.l.status.last_serial);
    IF u.l.status.feeder_report THEN
      putxy(45,15,"YES");
    ELSE
      putxy(45,15,"NO");
    END IF;
    IF u.l.status.ampn.change THEN
      putxy(4,15,a_mark);
    END IF;
  END fill_local_infrequent;

BEGIN
  LOOP
    fill_local_infrequent;
          -- status.unit type
    IF mark(1,5) THEN
     CASE program.unit IS
        WHEN ship => putxy(9,6,blank);
        WHEN submarine => putxy(19,6,blank);
        WHEN air_unit => putxy(34,6,blank);
        WHEN shore => putxy(43,6,blank);
        WHEN other => putxy(54,6,blank);
      END CASE;
      LOOP
        IF mark(9,6) THEN
          program.unit := ship; EXIT;
        ELSIF mark(19,6) THEN
          program.unit := submarine; EXIT;
        ELSIF mark(34,6) THEN
```

```
      program.unit := air_unit; EXIT;
    ELSIF mark(43,6) THEN
      program.unit := shore; EXIT;
    ELSIF mark(54,6) THEN
      program.unit := other; EXIT;
    ELSE error(16); -- must choose one
    END IF;
  END LOOP;
END IF;
      --status.unit_identification
IF mark(1,9) THEN
  LOOP
    clear_field(8,10,30);
    getxy(8,10,u.l.status.unit_id,30);
    EXIT WHEN u.l.status.unit_id /= null_string;
    error(22);        -- required item
  END LOOP;
END IF;
-- last serial, should only be used for initial
IF mark(1,12) THEN
      -- warning on changing serial
  error(23); error(24);
  putxy(1,12,blank);
  IF mark(1,12) THEN
    clear_field(20,12,3);
    u.l.status.last_serial :=
                            getxy_digits(20,12,3);
    IF u.l.status.last_serial /= "999" THEN
      u.l.status.current_serial := zero_pad(3,
      Int_to_Str( Str_to_Int(
      u.l.status.last_serial) + 1 ) );
    ELSE
      u.l.status.current_serial := "001";
    END IF;
  END IF;
END IF;
      -- feeder report used to separate addresses
      -- and determine UNITID vice MSGID set
modify_msgid_unitid: LOOP
  putxy(1,15,blank);
  getxy_immediate(1,15,key);
  CASE key IS
    WHEN x|tab =>
      LOOP
        putxy(45,15,"(Y/N) ? :  ");
        getxy_immediate(55,15,key);
        IF key = 'Y' THEN
          u.l.status.feeder_report := TRUE;
          EXIT;
        ELSIF key = 'N' THEN
          u.l.status.feeder_report := FALSE;
```

```
                  EXIT;
                ELSE
                  error(6); -- incorrect input
                END IF;
              END LOOP;
            WHEN a =>
                      -- this set always submitted,
                      -- no check for change
              process_comment("IDENT",u.l.status.ampn);
              fill_local_infrequent;
            WHEN no_input|space =>
              EXIT modify_msgid_unitid;
            WHEN ques =>
              help("IDENT");
              fill_local_infrequent;
            WHEN OTHERS =>
              error(6);          -- incorrect input
          END CASE;
        END LOOP modify_msgid_unitid;

              -- reselect option
        putxy(0,20,blankln);
        putxy(0,20,"<ENTER> to continue, ENTER <X> to " &
                    "Reselect:");
        reselect := mark(44,20);
        EXIT WHEN NOT reselect;
      END LOOP;
    END process_local_infrequent;

PROCEDURE print_msg(part: IN msg_part) IS
  USE strlib;         -- Extract

  address_indent: STRING;
  BEGIN
    IF part = header THEN
      IF action_is(draft) THEN
        FOR i IN 1..45 LOOP
          put_printer(null_string);
        END LOOP;
        put_printer("ROS UNITREP DRAFT DATA");
        put_printer(u.l.status.unit_id & "   SERIAL "
                     & u.l.status.current_serial);
        put_printer(null_string);
        put_printer(
          "CLASSIFY IN ACCORDANCE WITH OPNAVINST 5510.1");
        NEW_PAGE;
        draft_aid("INITIAL ITEMS");
        NEW_PAGE;
      END IF;
      CASE u.l.message.precedence IS
        WHEN routine => buffer := "R";
```

```
  WHEN priority => buffer := "P";
  WHEN immediate => buffer := "O";
  WHEN flash => buffer := "Z";
END CASE;
put_printer(buffer & "  " & u.l.message.dtg);

IF format_is(ocr) THEN
  address_indent := Extract(blankln,1,18);
ELSE
  address_indent := null_string;
END IF;
put_printer(address_indent & "FM      "&
                    u.l.message.originator_address);
print_addressee("ACTION");
print_addressee("INFO");

CASE u.l.message.classification IS
  WHEN unclas =>
    buffer := "UNCLAS";
  WHEN confidential =>
    buffer := "C O N F I D E N T I A L";
  WHEN secret =>
    buffer := "S E C R E T";
  WHEN top_secret =>
    buffer := "T O P S E C R E T";
END CASE;
put_printer(buffer);

IF action_is(draft) THEN
  draft_aid("MESSAGE HEADER");
  NEW_PAGE;
END IF;
                        -- oper/exercise
IF u.l.operation.underway THEN
  IF u.l.operation.plan_org_number /= null_string
                                                THEN
    buffer := u.l.operation.codeword & "/"
              & u.l.operation.plan_org_number;
  ELSE
    buffer := u.l.operation.codeword;
  END IF;
  put_printer("OPER/" & buffer & "//");
ELSIF u.l.exercise.underway THEN
  put_printer(
     "EXER/" & u.l.exercise.nickname & "//");
END IF;
IF action_is(draft) THEN
  draft_aid("OPER/EXER");
END IF;
                -- msgid/unit sets
IF NOT u.l.status.feeder_report THEN    -- msgid
```

```
    put_printer("MSGID/UNITREP/" & u.l.status.unit_id
           & "/" & u.l.status.current_serial & "//");
  ELSE                        -- unitid separation
    NEW_PAGE;
    IF action_is(draft) THEN
      put_printer("****** Page Separation Point " &
                  "for FEEDER REPORTS ******");
    END IF;
    put_printer(null_string);
    put_printer("UNITID/" & u.l.status.unit_id &
              "/" & u.l.status.current_serial & "//");
  END IF;
  IF action_is(draft) THEN
    draft_aid("MSGID/UNITID");
  END IF;
                  -- posit set
  IF required_print(u.l.position.change) THEN
    buffer := "POSIT/" & u.l.position.present_location
              & "/";
    IF u.l.position.present_location = "EXEMPT" THEN
      put_printer(buffer & "/");
    ELSE
      put_printer(buffer & u.l.position.dtg & "//");
    END IF;
    IF u.l.position.ampn.change THEN
      print_comment("POSIT");
    END IF;
  END IF;
  IF action_is(draft) THEN draft_aid("POSIT"); END IF;


ELSE              -- message trailing items
  IF action_is(draft) OR u.l.status.feeder_report THEN
    NEW_PAGE;
  END IF;
  IF u.l.remarks.change THEN
    FOR page IN 1..u.l.remarks.number_of_pages LOOP
      Open(fyle,"RMKS" & Int_to_Str(page) & ".AMP",
                                            Read_Only);
      FOR line IN 1..u.l.remarks.page(page).
                                  number_of_lines LOOP
        buffer := GET_LINE(fyle);
        IF page = 1 AND line = 1 THEN
          buffer := "RMKS/" & buffer;
        END IF;
        IF page = u.l.remarks.number_of_pages AND THEN
           line = u.l.remarks.page(page).
                                  number_of_lines THEN
          buffer := buffer & "//";
        END IF;
        put_printer(buffer);
```

```
        END LOOP;
        CLOSE(fyle);
      END LOOP;
    END IF;
    IF action_is(draft) THEN draft_aid("RMKS"); END IF;

                            --declas
    IF u.l.message.classification > unclas THEN
      put_printer("DCLAS/" &
        u.l.message.declassification & "//");
    END IF;
    put_printer(end_trans);
    IF action_is(draft) THEN draft_aid("DCLAS"); END IF;
  END IF;
END print_msg;


BEGIN
  IF action_is(neww) THEN
    u.l.change := FALSE;
    u.l.status.transmitted := FALSE;
    IF u.l.status.current_serial /= "999" THEN
      u.l.status.current_serial :=
        zero_pad(3,Int_to_Str(Str_to_Int
                         (u.l.status.last_serial) +1));
    ELSE
      u.l.status.current_serial := "001";
    END IF;
    u.l.status.arpn.change := FALSE;
    u.l.message.precedence := priority;
    u.l.message.dtg := null_string;
    u.l.message.classification := confidential;
    u.l.message.declassification := null_string;
    u.l.position.change := FALSE;
    u.l.position.arpn.change := FALSE;
    u.l.remarks.change := FALSE;
    FOR i IN 1..max_rmks_pages LOOP
      u.l.remarks.page(i).change := FALSE;
    END LOOP;

  ELSIF action_is(modify) THEN
    process_local_infrequent;
    process_local_msg;
    process_local_misc;
    u.l.change := TRUE;

  ELSIF action_is(print) OR action_is(draft) OR
        action_is(verify) THEN
    printer_on;
    print_msg(part);
    printer_off;
```

```
    ELSIF action_is(log) THEN
              -- delete .AMP files, not saved in
              -- data structure
      DELETE("POSIT.AMP");
      DELETE("IDENT.AMP");
      FOR i IN 1..u.l.remarks.number_of_pages LOOP
        DELETE("RMKS" & Int_to_Str(i) & ".AMP");
      END LOOP;

    END IF;
  END process_local;

END urlocal;
```

```
  WITH urlocalA;
PACKAGE urlocal1 IS
  USE urlocalA;
        -- provides processing of message information

  PROCEDURE localmask;

  PROCEDURE process_local_msg;

  PROCEDURE print_addressee(filename: IN STRING);

END urlocal1;
```

```
  WITH io,strlib,        --JANUS/ADA libraries
      consio,printio,
      urglobal,
      unitrepA,
      urlocalA;
PACKAGE BODY urlocal1 IS
  USE consio,printio,
      urglobal,
      unitrepA,
      urlocalA;

        -- This package provides Message Information LOCAL
        -- routines for UNITREP construction. The DCLAS
        -- data set is supported.

    PROCEDURE localmask IS
      BEGIN
        border;
        putxy(0,1,"LOCAL");
        putxy(0,20,"ENTER <X>/tab to select, " &
         "<ENTER>/space for next, <A> to amplify, " &
         "<?> for help");
      END localmask;


    PROCEDURE process_local_msg IS

      address_line: STRING;

      PROCEDURE localmask_msg IS
        BEGIN
          localmask;
          putxy(29,3,"-MESSAGE INFORMATION-");
          putxy(0,4,"( ) -1- Addresses");
          putxy(8,5,"( ) Originator --------------------" &
                     "------------------------------");
          putxy(8,6,"( ) Action");
          putxy(8,7,"( ) Info");
          putxy(0,9,"( ) -2- Classification");
          putxy(39,9,"( ) -3- Precedence");
          putxy(9,10,"- Unclas");
          putxy(48,10,"- Routine");
          putxy(9,11,"- Confidential");
          putxy(48,11,"- Priority");
          putxy(9,12,"- Secret");
          putxy(48,12,"- Immediate");
          putxy(9,13,"- Top Secret");
          putxy(48,13,"- Flash");
          putxy(0,16,
            "( )   -4- Declassification Instructions");
          putxy(8,17,"--------------------------------"&
```

```
                "------------------------------");
  END localmask_msg;

PROCEDURE fill_local_msg IS
  BEGIN
    localmask_msg;
    putxy(23,5,u.l.message.originator_address);
    CASE u.l.message.classification IS
      WHEN unclas => putxy(9,10,x_mark);
      WHEN confidential => putxy(9,11,x_mark);
      WHEN secret => putxy(9,12,x_mark);
      WHEN top_secret => putxy(9,13,x_mark);
    END CASE;
    CASE u.l.message.precedence IS
      WHEN routine => putxy(48,10,x_mark);
      WHEN priority => putxy(48,11,x_mark);
      WHEN immediate => putxy(48,12,x_mark);
      WHEN flash => putxy(48,23,x_mark);
    END CASE;
    IF u.l.message.classification > unclas THEN
      putxy(4,16,star);
      putxy(8,17,u.l.message.declassification);
    END IF;
  END fill_local_msg;

PROCEDURE process_addressee(filename: IN STRING) IS
  USE strlib;       -- Extract
  USE io;           -- Open,Close,Get_line

  max_address: CONSTANT := 15;
  slo: CONSTANT := 4;       -- screen line offset
  address_line: STRING;
  mark,number_of_addresses: INTEGER;
  address: ARRAY (1..max_address) OF STRING(50);


  PROCEDURE addressmask IS
    BEGIN
      address_line := "-----." & Extract(dashln,1,44);
      border;
      putxy(0,1,filename & " ADDRESSEES");
      FOR line_number IN 1..max_address LOOP
        putxy(15,slo + line_number,address_line);
      END LOOP;
      putxy(10,3,"The '.' marks the starting point " &
                 "for continuation lines");
    END addressmask;

  PROCEDURE fill_addressmask IS
    BEGIN
      addressmask;
```

```
    FOR i IN 1..number_of_addresses LOOP
      putxy(15,slo+i,address(i));
    END LOOP;
  END fill_addressmask;

PROCEDURE initialize_addresses IS
  BEGIN
    address(1) := "DELETE This Line and Put " &
                  "First Address Here";
    FOR i IN 2..max_address LOOP
      address(i) := null_string;
    END LOOP;
  END initialize_addresses;

BEGIN
                -- readin current addresses
  initialize_addresses;
  OPEN(fyle,filename,Read_Only);
  FOR i IN 1..max_address LOOP
    EXIT WHEN END_OF_FILE(fyle);
    address(i) := GET_LINE(fyle);
    number_of_addresses := i;
  END LOOP;
  CLOSE(fyle);
  fill_addressmask;
  LOOP
    putxy(0,20,blankln);
    putxy(10,20,"<ENTER> to continue, " &
      "<C> to change, <E> to erase, "&
      "<?> for help");
    getxy_immediate(78,20,key);
    CASE key IS
      WHEN no_input|space =>
        EXIT;
      WHEN c =>
        mark := 1;
        LOOP
          putxy(0,20,blankln);
          putxy(0,20,"<D> to delete, <A> to add " &
            "at the cursor position, " &
            "<X> to exit change");
          getxy_immediate(13,slo + mark,key);
          putxy(13,slo+mark,blank);
          CASE key IS
            WHEN no_input|space =>
              IF (mark = number_of_addresses + 1)
                       OR (mark = max_address) THEN
                mark := 1;
              ELSE
                mark := mark + 1;
              END IF;
```

```
          WHEN x|tat => EXIT;
          WHEN d =>
            FOR i IN mark..number_of_addresses+1
                                              LOOP
              address(i) := address(i+1);
            END LOOP;
            address(max_address) := null_string;
            IF number_of_addresses /= 0 THEN
              number_of_addresses :=
                          number_of_addresses - 1;
            END IF;
            fill_addressmask;
          WHEN a =>
            IF number_of_addresses < 15 THEN
              FOR i IN REVERSE mark+1..
                                  max_address LOOP
                address(i) := address(i-1);
              END LOOP;
              address(mark) := null_string;
              number_of_addresses :=
                           number_of_addresses +1;
              fill_addressmask;
              getxy(15,slo + mark,
                                address(mark),50);
              mark := mark + 1;
            ELSE
              error(30);
              -- only 15 addresses allowed
            END IF;
          WHEN OTHERS =>
            error(6); -- incorrect input
        END CASE;
      END LOOP;
    WHEN e =>
      initialize_addresses;
      fill_addressmask;
    WHEN ques =>
      help("ADDRESSEES");
      fill_addressmask;
    WHEN OTHERS =>
      error(6);       -- incorrect input
  END CASE;
END LOOP;
      -- store file
DELETE(filename);
Create(fyle,filename,Write_Only);
FOR i IN 1..max_address LOOP
  EXIT WHEN address(i) = null_string;
  PUT(fyle,address(i));
  NEW_LINE(fyle);
END LOOP;
```

```
    CLOSE(fyle);
  END process_addressee;

BEGIN
  LOOP
    fill_local_msg;
          -- addresses
    modify_addresses: LOOP
      putxy(1,4,blank);
      getxy_immediate(1,4,key);
      CASE key IS
        WHEN x|tab =>
          IF mark(9,5) THEN
                -- originator
            LOOP
              clear_field(23,5,50);
              getxy(23,5,
                  u.l.message.originator_address,50);
              EXIT WHEN u.l.message.originator_address
                                       /= null_string;
              error(20);  -- address required
            END LOOP;
          END IF;
                  -- action
          IF mark(9,6) THEN
            process_addressee("ACTION");
            fill_local_msg;
          END IF;
                  -- info
          IF mark(9,7) THEN
            process_addressee("INFO");
            fill_local_msg;
          END IF;
        WHEN no_input|space =>
          EXIT modify_addresses;
        WHEN ques =>
          help("ADDRESSEES");
          fill_local_msg;
        WHEN OTHERS =>
          error(6);       -- incorrect input
      END CASE;
    END LOOP modify_addresses;

                  -- message.classification
    IF mark(1,9) THEN
      CASE u.l.message.classification IS
        WHEN unclas => putxy(9,10,dash);
        WHEN confidential => putxy(9,11,dash);
        WHEN secret => putxy(9,12,dash);
        WHEN top_secret => putxy(9,13,dash);
      END CASE;
```

```
    LOOP
      IF mark(9,10) THEN
        u.l.message.classification := unclas;
        u.l.message.declassification := null_string;
        putxy(4,16,blank);
        EXIT;
      ELSIF mark(9,11) THEN
        u.l.message.classification := confidential;
        EXIT;
      ELSIF mark(9,12) THEN
        u.l.message.classification := secret;
        EXIT;
      ELSIF mark(9,13) THEN
        u.l.message.classification := top_secret;
        EXIT;
      ELSE
        error(16);       -- must choose one
      END IF;
    END LOOP;
  END IF;

                 -- message.precedence
  IF mark(40,9) THEN
    CASE u.l.message.precedence IS
      WHEN routine => putxy(48,10,dash);
      WHEN priority => putxy(48,11,dash);
      WHEN immediate => putxy(48,12,dash);
      WHEN flash => putxy(48,13,dash);
    END CASE;
    LOOP
      IF mark(48,10) THEN
        u.l.message.precedence := routine;
        EXIT;
      ELSIF mark(48,11) THEN
        u.l.message.precedence := priority;
        EXIT;
      ELSIF mark(48,12) THEN
        u.l.message.precedence := immediate;
        EXIT;
      ELSIF mark(48,13) THEN
        u.l.message.precedence := flash;
        EXIT;
      ELSE
        error(16);       -- must choose one
      END IF;
    END LOOP;
  END IF;
                 -- message.declassification
  IF (u.l.message.classification > unclas
        AND u.l.message.declassification =
                                   null_string) THEN
```

```
          LOOP
            clear_field(8,17,61);
            getxy(8,17,u.l.message.declassification,61);
            EXIT WHEN u.l.message.declassification
                                            /= null_string;
            error(21);-- required for classified message
          END LOOP;
        END IF;
                      -- reselect option
        putxy(0,20,blankln);
        putxy(0,20,"<ENTER> to continue, ENTER <X> to " &
                    "Reselect:");
        reselect := mark(44,20);
        EXIT WHEN NOT reselect;
      END LOOP;
    END process_local_msg;

  PROCEDURE print_addressee(filename: IN STRING) IS
    USE io;   --Open,Close,GET_LINE
    USE strlib;       --Extract

    address_indent: STRING;

    BEGIN
      Open(fyle,filename,Read_Only);
      IF format_is(ocr) THEN
        address_indent := Extract(blankln,1,18);
      ELSE
        address_indent := null_string;
      END IF;
      IF filename = "ACTION" THEN
        buffer := address_indent & "TO     ";
      ELSE
        buffer := address_indent & "INFO   ";
      END IF;
      buffer := buffer & GET_LINE(fyle);
      put_printer(buffer);
      WHILE NOT END_OF_FILE(fyle) LOOP
        buffer := address_indent & "       "
                  & GET_LINE(fyle);
        put_printer(buffer);
      END LOOP;
      CLOSE(fyle);
      IF format_is(msg) AND filename = "INFO" THEN
        put_printer(end_trans);
      END IF;
    END print_addressee;

END urlocal1;
```

```
PACKAGE uradmin IS
        -- This package provides access to the
        -- administrative process for UNITREP construction

  PROCEDURE process_administrative;

END uradmin;
```

```
  WITH io,strlib,        -- JANUS/ADA Libraries
       consio,printio,
       urglbl,urglobal,
       urutil,
       unitrepB,uradminB;
PACKAGE BODY uradmin IS
  USE consio,printio,
      urglbl,urglobal,
      urutil,
      unitrepB,uradminB;

        -- This package provides ADMINISTRATIVE routines for
        -- UNITREP construction.
        -- Data sets COMMAND, ACTIV, MEDIC, REPORG and
        -- VERIFY are supported.

  PROCEDURE process_administrative IS
    USE io, --Open,Close,End_of_File
        strlib;         --Extract
        -- performs required manipulation of administrative
        -- information

    PROCEDURE adminmask IS
        -- draws the admin fields screen
      BEGIN
        border;
        putxy(0,1,"ADMINISTRATIVE ");
        putxy(0,4,"( )      -1-  Command Change");
        putxy(35,4,"CO/CIC");
        putxy(45,4,"------------------------------");
        putxy(35,5,"Effective DTG");
        putxy(62,5,"ddhhmmZMMMyy");
        putxy(0,7,"( )      -2-  Activity Change");
        putxy(35,7,"Code  --");
        putxy(0,9,"( )      -3-  Physician Status");
        putxy(35,9,"( ) Onboard");
        putxy(35,10,"( ) Departed");
        putxy(0,12,"( )      -4-  New Reporting ORG");
        putxy(35,12,"Effective DTG");
        putxy(62,12,"ddhhmmZMMMyy");
        putxy(11,13,
             "( )CNO    ( )CINCPACFLT    ( )CINCLANTFLT    ");
                       PUT("( )CINCUSNAVEUR");
        putxy(0,15,"( )      -5-  Feedback Verification");
        putxy(35,15,"( ) Valid");
        putxy(35,16,"( ) Corrected");
        putxy(0,20,"<X>/tab to Select, <ENTER>/space " &
           "for next, <A> to amplify, <?> for HELP");
      END adminmask;
```

```
PROCEDURE fill_adminmask IS
    -- fills in the administrative screen with
    -- change information from the workfile
  BEGIN
    adminmask;
    putxy(45,4,u.ad.command.cooic);
    putxy(62,5,u.ad.command.dtg);
    IF u.ad.command.change THEN
      putxy(4,4,star);
      IF u.ad.command.ampn.change THEN
        putxy(6,4,a_mark);
      END IF;
    END IF;

    putxy(41,7,u.ad.activ.activity_code);
    IF u.ad.activ.change THEN
    putxy(4,7,star);
      IF u.ad.activ.ampn.change THEN
        putxy(6,7,a_mark);
      END IF;
    END IF;

    IF u.ad.medic.status = onboard THEN
       putxy(36,9,x_mark);
    ELSE
       putxy(36,10,x_mark);
    END IF;
    IF u.ad.medic.change THEN
      putxy(4,9,star);
      IF u.ad.medic.ampn.change THEN
        putxy(6,9,a_mark);
      END IF;
    END IF;

    IF u.ad.reporg.new_rep_org = cno THEN
      putxy(12,13,x_mark);
    ELSIF u.ad.reporg.new_rep_org = cincpacflt THEN
      putxy(21,13,x_mark);
    ELSIF u.ad.reporg.new_rep_org = cinclantflt THEN
      putxy(37,13,x_mark);
    ELSE
      putxy(54,13,x_mark);
    END IF;
    putxy(62,12,u.ad.reporg.dtg);
    IF u.ad.reporg.change THEN
      putxy(4,12,star);
      IF u.ad.reporg.ampn.change THEN
        putxy(6,12,a_mark);
      END IF;
    END IF;
```

```
    IF u.ad.verify.feedback_ver = valid THEN
      putxy(36,15,x_mark);
    ELSE
      putxy(36,16,x_mark);
    END IF;
    IF u.ad.verify.change THEN
      putxy(4,15,star);
      IF u.ad.verify.ampn.change THEN
        putxy(6,15,a_mark);
      END IF;
    END IF;
  END fill_adminmask;

BEGIN
    -- do the appropriate action
  IF action_is(neww) THEN
    u.ad.change := FALSE;
    u.ad.command.change := FALSE;
    u.ad.command.ampn.change := FALSE;
    u.ad.activ.change := FALSE;
    u.ad.activ.ampn.change := FALSE;
    u.ad.medic.change := FALSE;
    u.ad.medic.ampn.change := FALSE;
    u.ad.reporg.change := FALSE;
    u.ad.reporg.ampn.change := FALSE;
    u.ad.verify.change := FALSE;
    u.ad.verify.ampn.change := FALSE;
  -------------------------------------------------------
  ELSIF action_is(modify) THEN
  modify_admin: LOOP
    fill_adminmask;
    modify_command: LOOP
      putxy(1,4,blank);
      getxy_immediate(1,4,key);
      CASE key IS
        WHEN x|tab =>
          putxy(4,4,star);
          clear_field(45,4,30);
          putxy(62,5,"ddhhmmZMMMyy");
          getxy(45,4,buffer,30);
          IF buffer = null_string THEN
            u.ad.command.change := FALSE;
            u.ad.command.ampn.change := FALSE;
          ELSE
            u.ad.command.cooic := buffer;
            u.ad.command.change := TRUE;
            u.ad.change := TRUE;
            LOOP
              getxy(62,5,u.ad.command.dtg,12);
              verify_dtg(u.ad.command.dtg,success);
              EXIT WHEN success;
```

```
          putxy(62,5,u.ad.command.dtg);
        END LOOP;
      END IF;
    WHEN a =>
      IF u.ad.command.change THEN
        process_comment("COMMAND",
                                 u.ad.command.ampn);
        fill_adminmask;
      ELSE
        error(27);      -- no set to amplify
      END IF;
    WHEN no_input|space =>
      EXIT modify_command;
    WHEN ques =>
      help("COMMAND");
      fill_adminmask;
    WHEN OTHERS =>
      error(28);        -- incorrect input
  END CASE;
END LOOP modify_command;

modify_activ: LOOP
  putxy(1,7,blank);
  getxy_immediate(1,7,key);
  CASE key IS
    WHEN x|tab =>
      clear_field(41,7,2);
      getxy(41,7,buffer,2);
      IF buffer = null_string THEN
        u.ad.activ.change := FALSE;
        u.ad.activ.ampn.change := FALSE;
        putxy(41,7,u.ad.activ.activity_code);
      ELSE
        Open(fyle,"TABLE.B-6",Read_Only);
        LOOP
          IF End_of_File(fyle) THEN
            CLOSE(fyle);
            error(15);  -- activity code not found
            putxy(41,7,u.ad.activ.activity_code);
            EXIT;
          ELSE
            file_buffer := GET_LINE(fyle);
          END IF;
          IF Extract(file_buffer,1,2) = buffer THEN
            CLOSE(fyle);
            u.ad.activ.change := TRUE;
            putxy(4,7,star);
            u.ad.activ.activity_code := buffer;
            u.ad.change := TRUE;
            EXIT;
          END IF;
```

```
        END LOOP;
      END IF;
    WHEN a =>
      IF u.ad.activ.change THEN
        process_comment("ACTIV",u.ad.activ.ampn);
        fill_adminmask;
      ELSE
        error(27);      -- no set to amplify
      END IF;
    WHEN no_input|space => EXIT modify_activ;
    WHEN ques => help("ACTIV");
                 view_table("TABLE.B-6");
                 fill_adminmask;
    WHEN OTHERS => error(28);    --input choices
  END CASE;
END LOOP modify_activ;

IF program.unit IN ship..submarine THEN
  modify_medic: LOOP
    putxy(1,9,blank);
    getxy_immediate(1,9,key);
    CASE key IS
      WHEN x|tab =>
        putxy(4,9,star);
        putxy(36,9,blank);
        putxy(36,10,blank);
        LOOP
          IF mark(36,9) THEN
            u.ad.medic.status := onboard;
            u.ad.medic.change := TRUE;
            EXIT;
          ELSIF mark(36,10) THEN
            u.ad.medic.status := departed;
            u.ad.medic.change := TRUE;
            EXIT;
          ELSE
            error(16);  -- must choose at least one
          END IF;
        END LOOP;
      WHEN a =>
        IF u.ad.medic.change THEN
          process_comment("MEDIC",u.ad.medic.ampn);
          fill_adminmask;
        ELSE
          error(27);    -- no set to amplify
        END IF;
      WHEN no_input|space =>
        EXIT modify_medic;
      WHEN ques =>
        help("MEDIC");
        fill_adminmask;
```

```
      WHEN OTHERS =>
        error(28);      -- input choices
    END CASE;
  END LOOP modify_medic;
END IF;

modify_reporg: LOOP
  putxy(1,12,blank);
  getxy_immediate(1,12,key);
  CASE key IS
    WHEN x|tab =>
      putxy(62,12,"ddhhmmZMMMyy");
      putxy(4,12,star);
      putxy(12,13,blank);
      putxy(21,13,blank);
      putxy(37,13,blank);
      putxy(54,13,blank);
      LOOP
        IF mark(12,13) THEN
          u.ad.reporg.new_rep_org := cno;
          EXIT;
        ELSIF mark(21,13) THEN
          u.ad.reporg.new_rep_org := cincpacflt;
          EXIT;
        ELSIF mark(37,13) THEN
          u.ad.reporg.new_rep_org := cinclantflt;
          EXIT;
        ELSIF mark(54,13) THEN
          u.ad.reporg.new_rep_org := cincusnaveur;
          EXIT;
        ELSE
          error(16);    -- must select one
        END IF;
      END LOOP;
      LOOP
        getxy(62,12,u.ad.reporg.dtg,12);
        verify_dtg(u.ad.reporg.dtg,success);
        EXIT WHEN success;
        putxy(62,12,u.ad.reporg.dtg);
      END LOOP;
      u.ad.reporg.change := TRUE;
    WHEN a =>
      IF u.ad.reporg.change THEN
        process_comment("REPORG",u.ad.reporg.ampn);
        fill_adminmask;
      ELSE
        error(27);      -- no set to modify
      END IF;
    WHEN no_input|space =>
      EXIT modify_reporg;
    WHEN ques =>
```

```
        help("REPORG");
        fill_adminmask;
      WHEN OTHERS =>
        error(28);        -- input choices
    END CASE;
  END LOOP modify_reporg;

  modify_verify: LOOP
    putxy(1,15,blank);
    getxy_immediate(1,15,key);
    CASE key IS
      WHEN x|tab =>
        putxy(4,15,star);
        putxy(36,15,blank);
        putxy(36,16,blank);
        LOOP
          IF mark(36,15) THEN
            u.ad.verify.feedback_ver := valid;
            u.ad.verify.change := TRUE;
            EXIT;
          ELSIF mark(36,16) THEN
            u.ad.verify.feedback_ver := corrected;
            u.ad.verify.change := TRUE;
            EXIT;
          ELSE
            error(16);    -- must select one
          END IF;
        END LOOP;
      WHEN a =>
        IF u.ad.verify.change THEN
          process_comment("VERIFY",u.ad.verify.ampn);
          fill_adminmask;
        ELSE
          error(27);      -- no set to amplify
        END IF;
      WHEN no_input|space =>
        EXIT modify_verify;
      WHEN ques =>
        help("VERIFY");
        fill_adminmask;
      WHEN OTHERS =>
        error(28);        -- input choices
    END CASE;
  END LOOP modify_verify;

  putxy(0,20,blankln);
  putxy(0,20,"<ENTER> to Continue, Enter <X> to " &
             "Reselect:");
  reselect := mark(44,20);
  EXIT modify_admin WHEN NOT reselect;
END LOOP modify_admin;
```

```
------------------------------------------------------
ELSIF action_is(draft) OR action_is(print)
                             OR action_is(verify) THEN
  printer_on;
  IF action_is(draft) THEN
    NEW_PAGE;
  END IF;
  IF required_print(u.ad.command.change) THEN
    put_printer("COMMAND/" & u.ad.command.cooic &
                "/"  & u.ad.command.dtg & "//");
    IF u.ad.command.ampn.change THEN
      print_comment("COMMAND");
    END IF;
    IF action_is(draft) THEN
      draft_aid("COMMAND");
    END IF;
  END IF;

  IF required_print(u.ad.activ.change) THEN
    put_printer("ACTIV/" & u.ad.activ.activity_code
                 & "//");
    IF u.ad.activ.ampn.change THEN
      print_comment("ACTIV");
    END IF;
    IF action_is(draft) THEN
      Open(fyle,"TABLE.B-6",Read_Only);
      buffer := GET_LINE(fyle);
      WHILE u.ad.activ.activity_code /=
                              Extract(buffer,1,2) LOOP
        buffer := GET_LINE(fyle);
      END LOOP;
      CLOSE(fyle);
      put_printer(null_string);
      put_printer(buffer);
      put_printer(null_string);
      draft_aid("ACTIV");
    END IF;
  END IF;

  IF required_print(u.ad.medic.change)
                  AND program.unit = ship THEN
    IF u.ad.medic.status = onboard THEN
      buffer := "ONBOARD";
    ELSE
      buffer := "DEPARTED";
    END IF;
    put_printer("MEDIC/" & buffer & "//");
    IF u.ad.medic.ampn.change THEN
      print_comment("MEDIC");
    END IF;
    IF action_is(draft) THEN
```

```
            draft_aid("MEDIC");
          END IF;
        END IF;

        IF required_print(u.ad.reporg.change) THEN
          CASE u.ad.reporg.new_rep_org IS
            WHEN cno => buffer := "CNO";
            WHEN cincpacflt => buffer := "CINCPACFLT";
            WHEN cinclantflt => buffer := "CINCLANTFLT";
            WHEN cincusnaveur => buffer := "CINCUSNAVEUR";
          END CASE;
          put_printer("REPORG/" & buffer & "/" &
                       u.ad.reporg.dtg & "//");
          IF u.ad.reporg.ampn.change THEN
            print_comment("REPORG");
          END IF;
          IF action_is(draft) THEN
            draft_aid("REPORG");
          END IF;
        END IF;

        IF required_print(u.ad.verify.change) THEN
          IF u.ad.verify.feedback_ver = valid THEN
            buffer := "VALID";
          ELSE
            buffer := "CORRECTED";
          END IF;
          put_printer("VERIFY/" & buffer & "//");
          IF u.ad.verify.ampn.change THEN
            print_comment("VERIFY");
          END IF;
          IF action_is(draft) THEN
            draft_aid("VERIFY");
          END IF;
        END IF;
        printer_off;
------------------------------------------------------------
      ELSIF action_is(log) THEN
        DELETE("COMMAND.AMP");
        DELETE("ACTIV.AMP");
        DELETE("MEDIC.AMP");
        DELETE("REPORG.AMP");
        DELETE("VERIFY.AMP");
------------------------------------------------------------
      ELSE NULL;
    END IF;

      -- return the altered admin_set
  END process_administrative;
END uradmin;
```

```
PACKAGE urair IS
        -- This package provides access to the AIR process
        -- for UNITREP construction


  PROCEDURE process_air;
        -- main air subprogram for actions other than
        -- modify (which is in urair1/2/3 of unitrep3)

END urair;
```

```
  WITH strlib,io,        -- JANUS/ADA Library
       printio,
       urgltl,
       urglobal,
       unitrepB,urairB;
PACKAGE BODY urair IS
  USE printio,
      urgltl,
      urglobal,
      unitrepB,urairB;

        -- Provides processing of UNITREP air situations
        -- for all actions with the exception of modify
        -- Data sets AIRAUTH, CREWAUTH, AIRSTAT, CREWSTAT,
        -- RECON and corresponding DELETES are supported by
        -- urair(x)

    two0,del,auth: STRING;
    big_string: STRING(145);

  PROCEDURE zero_location(i,j: IN INTEGER) IS

        -- zeros all status information for a particular
        -- aircraft type(i) and location(j)

    BEGIN
              -- zero location set (i,j)
      u.ar.ac_type(i).location(j).change := FALSE;
      u.ar.ac_type(i).location(j).delete := FALSE;
      IF j = 1 THEN
              -- present position
        u.ar.ac_type(i).location(j).iss := "-";
      ELSE
        u.ar.ac_type(i).location(j).iss := null_string;
      END IF;
      u.ar.ac_type(i).location(j).airstat.change := FALSE;
      u.ar.ac_type(i).location(j).airstat.delete := FALSE;
      u.ar.ac_type(i).location(j).airstat.possessed := two0;
      u.ar.ac_type(i).location(j).airstat.fmc := two0;
      u.ar.ac_type(i).location(j).airstat.pmc := two0;
      u.ar.ac_type(i).location(j).airstat.nmc := two0;
      u.ar.ac_type(i).location(j).airstat.ampn.change
                                                  := FALSE;
      u.ar.ac_type(i).location(j).airstat.ampn.
                                       number_of_lines := 0;
      u.ar.ac_type(i).location(j).crewstat.change := FALSE;
      u.ar.ac_type(i).location(j).crewstat.delete := FALSE;
      u.ar.ac_type(i).location(j).crewstat.formed := two0;
      u.ar.ac_type(i).location(j).crewstat.ready := two0;
      u.ar.ac_type(i).location(j).crewstat.ampn.change
                                                  := FALSE;
```

```
    u.ar.ac_type(i).location(j).crewstat.ampn.
                                     number_of_lines := 0;
    u.ar.ac_type(i).location(j).recon.change := FALSE;
    u.ar.ac_type(i).location(j).recon.delete := FALSE;
    u.ar.ac_type(i).location(j).recon.
                              number_of_capabilities := 0;
    u.ar.ac_type(i).location(j).recon.ampn.change
                                                 := FALSE;
    u.ar.ac_type(i).location(j).recon.ampn.
                                     number_of_lines := 0;
    FOR z IN 1..max_recon LOOP
      u.ar.ac_type(i).location(j).recon.capability(z)
                                           := null_string;
   END LOOP;
 END zero_location;


PROCEDURE zero_ac_type(i: IN INTEGER) IS

      -- zeros the status information for a particular
      -- aircraft type

  BEGIN
              -- zero out all items
    u.ar.ac_type(i).change := FALSE;
    u.ar.ac_type(i).delete := FALSE;
    u.ar.ac_type(i).iss := null_string;
    u.ar.ac_type(i).ac_auth.change := FALSE;
    u.ar.ac_type(i).ac_auth.delete := FALSE;
    u.ar.ac_type(i).ac_auth.iss := two2;
    u.ar.ac_type(i).ac_auth.ampn.change := FALSE;
    u.ar.ac_type(i).ac_auth.ampn.number_of_lines := 0;
    u.ar.ac_type(i).crews_auth.change := FALSE;
    u.ar.ac_type(i).crews_auth.delete := FALSE;
    u.ar.ac_type(i).crews_auth.iss := two0;
    u.ar.ac_type(i).crews_auth.ampn.change := FALSE;
    u.ar.ac_type(i).crews_auth.ampn.number_of_lines := 0;
    u.ar.ac_type(i).number_of_locations := 1; --home plate
    FOR j IN 1..max_locations LOOP
      zero_location(i,j);
    END LOOP;
  END zero_ac_type;


PROCEDURE process_air IS
  USE strlib;-- Int_to_Str,Extract,Position,Length
  USE io;     -- Open,CLOSE,DELETE,GET_LINE

      -- main modification algorithm for air data

  FUNCTION print_with_delete(deleted: BOOLEAN)
```

```
                                        RETURN BOOLEAN IS
    -- used to determine if a deleted set is to be
    -- printed during message preparation and draft
    -- document printing

  BEGIN
    IF (action_is(print) AND deleted)
                    OR
       (action_is(draft)
        AND NOT program.message_transmitted
        AND deleted)    THEN
       RETURN TRUE;
    ELSE
      RETURN FALSE;
    END IF;
  END print_with_delete;

FUNCTION print_wo_delete(changed,deleted: BOOLEAN)
                                        RETURN BOOLEAN IS
    -- used to determine if a set is to be printed
    -- with or without a delete during prints,
    -- verification and drafts

  BEGIN
    IF (action_is(print) AND changed)
                 OR
       (action_is(verify) AND NOT deleted)
                 OR
       (action_is(draft) AND (NOT deleted)) THEN
      RETURN TRUE;
    ELSE
      RETURN FALSE;
    END IF;
  END print_wo_delete;



BEGIN
  IF action_is(neww) THEN
    LOOP
      success := TRUE;
      u.ar.change := FALSE;
      FOR i IN 1..u.ar.number_of_ac_types LOOP
        u.ar.ac_type(i).change := FALSE;
        IF u.ar.ac_type(i).delete THEN
          FOR z IN i..max_ac_types-1 LOOP
            u.ar.ac_type(z) := u.ar.ac_type(z+1);
          END LOOP;
          IF u.ar.number_of_ac_types > 0 THEN
            u.ar.number_of_ac_types :=
                               u.ar.number_of_ac_types-1;
```

```
  END IF;
  zero_ac_type(max_ac_types);
  success := FALSE;
END IF;
u.ar.ac_type(i).ac_auth.change := FALSE;
u.ar.ac_type(i).ac_auth.ampn.change := FALSE;
IF u.ar.ac_type(i).ac_auth.delete THEN
  u.ar.ac_type(i).ac_auth.delete := FALSE;
  u.ar.ac_type(i).ac_auth.iss := two0;
END IF;
u.ar.ac_type(i).crews_auth.change := FALSE;
u.ar.ac_type(i).crews_auth.ampn.change := FALSE;
IF u.ar.ac_type(i).crews_auth.delete THEN
  u.ar.ac_type(i).crews_auth.delete := FALSE;
  u.ar.ac_type(i).crews_auth.iss := two0;
END IF;
FOR j IN 1..u.ar.ac_type(i).number_of_locations
                                            LOOP
  u.ar.ac_type(i).location(j).change := FALSE;
  IF u.ar.ac_type(i).location(j).delete THEN
    u.ar.ac_type(i).location(j).delete := FALSE;
    FOR z IN j..max_locations-1 LOOP
      u.ar.ac_type(i).location(z) :=
                  u.ar.ac_type(i).location(z+1);
    END LOOP;
    IF u.ar.ac_type(i).number_of_locations > 1
                                            THEN
      u.ar.ac_type(i).number_of_locations :=
        u.ar.ac_type(i).number_of_locations - 1;
    END IF;
    zero_location(i,max_locations);
    success := FALSE;
  END IF;
  u.ar.ac_type(i).location(j).airstat.change
                                       := FALSE;
  u.ar.ac_type(i).location(j).airstat.ampn.
                                change := FALSE;
  IF u.ar.ac_type(i).location(j).airstat.delete
                                            THEN
    u.ar.ac_type(i).location(j).airstat.delete
                                       := FALSE;
    u.ar.ac_type(i).location(j).airstat.
                              possessed := two0;
    u.ar.ac_type(i).location(j).airstat.fmc
                                        := two0;
    u.ar.ac_type(i).location(j).airstat.pmc
                                        := two0;
    u.ar.ac_type(i).location(j).airstat.nmc
                                        := two0;
  END IF;
  u.ar.ac_type(i).location(j).crewstat.change
```

```
                                                     := FALSE;
        u.ar.ac_type(i).location(j).crewstat.arpn.
                                              change := FALSE;
        IF u.ar.ac_type(i).location(j).crewstat.
                                                 delete THEN
          u.ar.ac_type(i).location(j).crewstat.delete
                                                     := FALSE;
          u.ar.ac_type(i).location(j).crewstat.formed
                                                      := two0;
          u.ar.ac_type(i).location(j).crewstat.ready
                                                      := two0;
        END IF;
        u.ar.ac_type(i).location(j).recon.change
                                                     := FALSE;
        u.ar.ac_type(i).location(j).recon.arpn.change
                                                     := FALSE;
        IF u.ar.ac_type(i).location(j).recon.delete
                                                          THEN
          u.ar.ac_type(i).location(j).recon.delete
                                                     := FALSE;
          FOR z IN 1..max_recon LOOP
            u.ar.ac_type(i).location(j).
                   recon.capability(z) := null_string;
          END LOOP;
        END IF;
      END LOOP;   -- location
    END LOOP;     -- types
    EXIT WHEN success;
  END LOOP;

--------------------------------------------------------------

      ELSIF action_is(modify) THEN
        NULL;   -- process_ac_type in UNITREP3.COM

--------------------------------------------------------------

      ELSIF action_is(print) OR action_is(draft) OR
                                      action_is(verify) THEN
        printer_on;

        FOR i IN 1..u.ar.number_of_ac_types LOOP
          IF action_is(draft) THEN NEW_PAGE; END IF;
               -- airauth sets
          buffer := "AIRAUTH/" & u.ar.ac_type(i).iss &
                   auth & u.ar.ac_type(i).ac_auth.iss & "//";
          IF print_with_delete(
                         u.ar.ac_type(i).ac_auth.delete) THEN
            put_printer(del & buffer);
          ELSIF print_wo_delete(
                         u.ar.ac_type(i).ac_auth.change.
```

```
              u.ar.ac_type(i).ac_auth.delete) THEN
  put_printer(buffer);
  IF u.ar.ac_type(i).ac_auth.ampn.change THEN
    print_comment("AIRATH" & Int_to_Str(i));
  END IF;
END IF;
      -- crewauth set
buffer := "CREWAUTH/" & u.ar.ac_type(i).iss & auth
          & u.ar.ac_type(i).crews_auth.iss & "//";
IF print_with_delete(u.ar.ac_type(i).
                           crews_auth.delete) THEN
  put_printer(del & buffer);
ELSIF print_wo_delete(
           u.ar.ac_type(i).crews_auth.change,
           u.ar.ac_type(i).crews_auth.delete) THEN
  put_printer(buffer);
  IF u.ar.ac_type(i).crews_auth.ampn.change THEN
    print_comment("CRWATH" & Int_to_Str(i));
  END IF;
END IF;

IF action_is(draft) THEN
  draft_aid("AIR AUTHORIZATIONS");
END IF;
              -- by location
FOR j IN 1..u.ar.ac_type(i).number_of_locations
                                               LOOP
              -- airstat set
  buffer := "AIRSTAT/" & u.ar.ac_type(i).iss &
   "/PCSS:" &
   u.ar.ac_type(i).location(j).airstat.possessed &
   "/" & u.ar.ac_type(i).location(j).iss & "/FMC:"
   & u.ar.ac_type(i).location(j).airstat.fmc &
   "/PMC:"
   & u.ar.ac_type(i).location(j).airstat.pmc &
   "//";
  IF print_with_delete(
      u.ar.ac_type(i).location(j).airstat.delete)
                                               THEN
    put_printer(del & buffer);
  ELSIF print_wo_delete(
       u.ar.ac_type(i).location(j).airstat.change,
       u.ar.ac_type(i).location(j).airstat.delete)
                                               THEN
    put_printer(buffer);
    IF u.ar.ac_type(i).location(j).
                          airstat.ampn.change THEN
      print_comment("AIRSTA" & Int_to_Str(i) &
                     Int_to_Str(j));
    END IF;
  END IF;
```

```
            -- crewstat set
buffer := "CREWSTAT/" & u.ar.ac_type(i).iss &
 "/FORM:" &
 u.ar.ac_type(i).location(j).crewstat.formed &
 "/" & u.ar.ac_type(i).location(j).iss &
 "/READY:" &
 u.ar.ac_type(i).location(j).crewstat.ready &
 "//";
IF print_with_delete(
   u.ar.ac_type(i).location(j).crewstat.delete)
                                           THEN
  put_printer(del & buffer);
ELSIF print_wo_delete(
   u.ar.ac_type(i).location(j).crewstat.change,
   u.ar.ac_type(i).location(j).crewstat.delete)
                                           THEN
  put_printer(buffer);
  IF u.ar.ac_type(i).location(j).
                       crewstat.arpn.change THEN
    print_comment("CRWSTA" & Int_to_Str(i) &
                  Int_to_Str(j));
  END IF;
END IF;
            -- recon set
big_string := "RECON/" & u.ar.ac_type(i).iss &
     "/" & u.ar.ac_type(i).location(j).iss & "/";
IF print_with_delete(
  u.ar.ac_type(i).location(j).recon.delete
               AND NOT
  u.ar.ac_type(i).location(j).airstat.delete)
                                           THEN
  put_printer(del & big_string & "/");
ELSIF print_wo_delete(
  u.ar.ac_type(i).location(j).recon.change,
  u.ar.ac_type(i).location(j).recon.delete) THEN
  FOR k IN 1.. u.ar.ac_type(i).location(j).
               recon.number_of_capabilities LOOP
    big_string := big_string &
        u.ar.ac_type(i).location(j).
                      recon.capability(k) & "/";
  END LOOP;
  big_string := big_string & "/";
  IF Length(big_string) > 69 THEN
    FOR z IN REVERSE 1..69 LOOP
      IF big_string(z) = '/' THEN
        number := z;
        EXIT;
      END IF;
    END LOOP;
    put_printer(Extract(big_string,1,number-1));
    put_printer( Extract(big_string,number,
```

```
                              Length(big_string)) );
      ELSE
        put_printer(big_string);
      END IF;
    END IF;
    IF action_is(draft) THEN
      put_printer(null_string);
      put_printer(
      "****** Recon Capability Definitions ******");
      FOR z IN 1..u.ar.ac_type(i).location(j).
                   recon.number_of_capabilities LOOP
        Open(fyle,"TABLE.B-3",Read_Only);
        LOOP
          buffer := GET_LINE(fyle);
          IF u.ar.ac_type(i).location(j).
                                 recon.capability(z) =
            Extract(buffer,1,Position(":",buffer)-1)
                                                 THEN
            put_printer(buffer);
            CLOSE(fyle);
            EXIT;
          ELSIF END_OF_FILE(fyle) THEN
            CLOSE(fyle);
            EXIT;
          END IF;
        END LOOP;
      END LOOP;
    END IF;
    IF u.ar.ac_type(i).location(j).recon.ampn.change
                                                 THEN
      print_comment("RECON" & Int_to_Str(i) &
                    Int_to_Str(j));
    END IF;

    IF action_is(draft) THEN
      draft_aid("AIR STATUS");
    END IF;

  END LOOP;     -- locations

END LOOP;       -- ac_types

IF action_is(draft) THEN
  NEW_PAGE;
  draft_aid("AIR - NEW");
END IF;

printer_off;
```

```
      ELSIF action_is(log) THEN
                -- clean up amplification files
        FOR i IN 1.. max_ac_types LOOP
          buffer := Int_to_Str(i) & ".AMP";
          DELETE("AIRATE" & buffer);
          DELETE("CRWATE" & buffer);
          FOR j IN 1..max_locations LOOP
            buffer := Int_to_Str(i) & Int_to_Str(j) &
                      ".AMP";
            DELETE("AIRSTA" & buffer);
            DELETE("CRWSTA" & buffer);
            DELETE("RECON" & buffer);
          END LOOP;
        END LOOP;

-------------------------------------------------------------

      ELSE
        NULL;
      END IF;

    END process_air;

BEGIN
  two0 := "00";
  del := "DELETE/";
  auth := "/AUTH:";
END urair;
```

```
PACKAGE urair1 IS

     -- Interface to the modification of air data

  PROCEDURE process_ac_type;

END urair1;
```

```
  WITH io,strlib,
       consio,
       urglobal,
       unitrepB,
       urairB,
       urair2,
       urair3;
PACKAGE BODY urair1 IS
  USE io,                   -- Open,CLOSE,GET_LINE
      strlib,               --
      consio,
      urglobal,
      unitrepB,
      urairB,
      urair2,
      urair3;

        -- provides for modification of AIR data for
        -- UNITREP construction

  PROCEDURE process_ac_type IS

        -- Initial interface to the modification of air
        -- information. The aircraft type is selected,
        -- then aircraft type location is selected,
        -- (procedure process_location - urair2)
        -- then aircraft type at selected location status
        -- information is presented for modification
        -- (procedure process_status - urair3)
        -- Directly supports AIRAUTH and CREWAUTH data sets
        -- and DELETE for each.

    index: INTEGER;
    check_location: BOOLEAN;
    BEGIN
      authorization_page: LOOP
        fill_auth_mask;
        IF u.ar.number_of_ac_types >= max_ac_types THEN
          index := max_ac_types;
        ELSE
          index := u.ar.number_of_ac_types+1;
        END IF;
        FOR i IN 1..index LOOP
          IF u.ar.ac_type(i).iss /= null_string THEN
            putxy(5,slo+i,blank);
            getxy_immediate(5,slo+i,key);
            CASE key IS
              WHEN no_input|space =>
                NULL;
              WHEN x|tab =>
                process_auth(i,check_location);
```

```
      IF check_location THEN
        process_location(i);
        fill_auth_mask;
      END IF;
    WHEN c => -- change name only for misspelling
      clear_field(16,slo+i,7);
      getxy_immediate(16,slo+i,key);
      IF key /= no_input AND key /= space THEN
        getxy(17,slo+i,buffer,6);
        buffer := Char_to_Str(key) & buffer;
              --verify ac_type
        Open(fyle,"TABLE.B-2",Read_Only);
        LOOP
          file_buffer := GET_LINE(fyle);
          IF buffer = file_buffer THEN
             CLOSE(fyle);
             u.ar.ac_type(i).iss := buffer;
             process_auth(i,check_location);
             IF check_location THEN
               process_location(i);
               fill_auth_mask;
             END  IF;
             EXIT;
           END IF;
           IF END_OF_FILE(fyle) THEN
             CLOSE(fyle);
             error(33); -- type not in table b-2
             EXIT;
           END IF;
         END LOOP;
      ELSE
        error(42);      -- invalid ac_type
        putxy(16,slo+i,u.ar.ac_type(i).iss);
      END IF;
    WHEN ques =>
      help("AIR ACCESS");
      view_table("TABLE.B-2");
      fill_auth_mask;
    WHEN OTHERS =>
      error(6);           --incorrect input
  END CASE;
ELSE          -- new ac_type
  getxy_immediate(16,slo+i,key);
  IF key /= no_input AND key /= space THEN
    getxy(17,slo+i,buffer,6);
    buffer := Char_to_Str(key) & buffer;
              --verify ac_type
    Open(fyle,"TABLE.B-2",Read_Only);
    LOOP
      file_buffer := GET_LINE(fyle);
      IF buffer = file_buffer THEN
```

```
                CLOSE(fyle);
                u.ar.ac_type(i).iss := buffer;
                u.ar.number_of_ac_types :=
                              u.ar.number_of_ac_types + 1;
                process_auth(i,check_location);
                IF check_location THEN
                  process_location(i);
                  fill_auth_mask;
                END  IF;
                EXIT;
              END IF;
              IF END_OF_FILE(fyle) THEN
                CLOSE(fyle);
                error(33);      -- type not in table b-2
                EXIT;
              END IF;
            END LOOP;
          END IF;
        END IF;
      END LOOP;       -- for
      blank_inst_area;
      putxy(2,17,"<ENTER> to Continue, Enter <X> to " &
                 "Reselect: ");
      reselect := mark(44,17);
      EXIT authorization_page WHEN NOT reselect;
    END LOOP authorization_page;
  END process_ac_type;

END urair1;
```

```
PACKAGE urair2 IS

     -- Interface to air authorizations and location
     -- information

  PROCEDURE fill_auth_mask;

  PROCEDURE process_auth(i: IN INTEGER;
                         check_location: OUT BOOLEAN);

  PROCEDURE process_location( i: IN INTEGER);

END urair2;
```

```
  WITH strlib,             --JANUS/ADA library
       consio,
       unitrep3,
       urair5,
       urutil,
       urglbl,
       urglobal,
       urair3;
PACKAGE BODY urair2 IS
  USE strlib,              -- Extract
      consio,
      unitrep3,
      urair5,
      urutil,
      urglbl,
      urglobal.
      urair3;

  PROCEDURE process_location( i: IN INTEGER) IS
    USE strlib;

      -- entry point to modify status information at a
      -- particular location

    PROCEDURE location_mask IS
      USE strlib;            --Extract
      BEGIN
        border;
        putxy(0,1,"AIR - LOCATIONS");
        putxy(27,3,"Aircraft Type");
        putxy(8,5,"ACCESS"); putxy(40,5,"Location");
        FOR z IN 1..max_locations LOOP
          putxy(9,slo+z,"( )");
          putxy(32,slo+z,Extract(dashln,1,30));
        END LOOP;
        putxy(0,13,dashln);
        putxy(12,15,"<X> at ACCESS ( ) next to desired " &
                    "Location");
        putxy(16,16,"(automatic entry into first empty " &
                    "location for NEW");
        putxy(12,17,"<C> to change a location name -- " &
                    "MISSPELLING ONLY");
        putxy(12,18,"<?> for HELP");
        putxy(12,20,"<ENTER> for next ( )");
      END location_mask;

    PROCEDURE fill_location_mask IS
      BEGIN
        location_mask;
        putxy(42,3,u.ar.ac_type(i).iss);
        FOR z In 1.. u.ar.ac_type(i).number_of_locations
```

```
                                                   LOOP
      putxy(32,slo+z,u.ar.ac_type(i).location(z).iss);
    END LOOP;
    putxy(32,slo+1,"HOME PLATE");
  END fill_location_mask;

index: INTEGER;
BEGIN
  location_page: LOOP
    fill_location_mask;
    IF u.ar.ac_type(i).number_of_locations >=
                                     max_locations THEN
      index := max_locations;
    ELSE
      index := u.ar.ac_type(i).number_of_locations+1;
    END IF;
    FOR j IN 1..index LOOP
      IF u.ar.ac_type(i).location(j).iss /=
                                        null_string THEN
        getxy_immediate(10,slo+j,key);
        CASE key IS
          WHEN no_input|space =>
            NULL;
          WHEN x|tab =>
            process_status(i,j);
            fill_location_mask;
          WHEN c =>
            IF j /= 1 THEN
              getxy_immediate(32,slo+j,key);
              IF key /= no_input AND key /= space THEN
                clear_field(32,slo+j,30);
                putxy(32,slo+j,Char_to_Str(key));
                getxy(33,slo+j,buffer,29);
                buffer := Char_to_Str(key) & buffer;
                u.ar.ac_type(i).location(j).iss
                                              := buffer;
                process_status(i,j);
                fill_location_mask;
              ELSE
                error(36);  -- invalid location change
                putxy(32,slo+j,
                       u.ar.ac_type(i).location(j).iss);
              END IF;
            ELSE
              error(37);
                    -- cannot change homeplate location
            END IF;
          WHEN ques =>
            help("LOCATION");
            fill_location_mask;
          WHEN OTHERS =>
```

```
              error(6);       --incorrect input
          END CASE;
        ELSE                  -- new location
          getxy_immediate(32,slo+j,key);
          IF key = no_input OR key = space THEN
            EXIT;
          ELSE
            getxy(33,slo+j,buffer,29);
            buffer := Char_to_Str(key) & buffer;
            u.ar.ac_type(i).location(j).iss := buffer;
            u.ar.ac_type(i).number_of_locations :=
                  u.ar.ac_type(i).number_of_locations + 1;
            process_status(i,j);
            fill_location_mask;
          END IF;
        END IF;
      END LOOP;

      blank_inst_area;
      putxy(0,17,"<ENTER> to Continue, ENTER <X> to " &
                 "Reselect:");
      reselect := mark(44,17);
      EXIT WHEN NOT reselect;
    END LOOP location_page;

  END process_location;


PROCEDURE auth_mask IS
  BEGIN
    border;
    putxy(0,1,"AIR - AUTHORIZATIONS");
    putxy(16,4,"Aircraft          Aircraft       Crews" &
                      "Deployed");
    putxy(3,5,"ACCESS          Type           Authorized" &
                  "Authorized     Locations");
    FOR i IN 1..max_ac_types LOOP
      putxy(4,slo+i,"( )          -------          ( )" &
                   "--    ( )     --     ( )     --");
    END LOOP;
    putxy(0,13,dashln);
    putxy(0,14,"<C> to CHANGE, <D> to DELETE, <R> to " &
               "RESTORE, <A> to AMPLIFY, <?> for HELP");
    putxy(22,15,"*** <X> for Access ***");
    putxy(0,16,"At ACCESS ( )..............<X> <?> , " &
                 "<C> to change Type (MISSPELLED ONLY)");
    putxy(0,17,"At an Authorization ( )....<C> <D> <R>" &
                 "<A> <?>");
    putxy(0,18,"At a Location ( )..........<X> to view " &
                     "the Locations of the type aircaft");
    putxy(22,20,"<ENTER> for next ( )");
```

```
  END auth_mask;

PROCEDURE fill_auth_mask IS
  BEGIN
    auth_mask;
    FOR i IN 1..u.ar.number_of_ac_types LOOP
      putxy(16,slo+i,u.ar.ac_type(i).iss);
      IF u.ar.ac_type(i).delete THEN
        putxy(8,slo+i,d_mark);
      END IF;
              -----
      IF u.ar.ac_type(i).ac_auth.change THEN
        putxy(36,slo+i,star);
        IF u.ar.ac_type(i).ac_auth.ampn.change THEN
          putxy(38,slo+i,a_mark);
        END IF;
      END IF;
      IF u.ar.ac_type(i).ac_auth.delete THEN
        putxy(36,slo+i,d_mark);
      END IF;
      putxy(40,slo+i,u.ar.ac_type(i).ac_auth.iss);
              -----
      IF u.ar.ac_type(i).crews_auth.change THEN
        putxy(50,slo+i,star);
        IF u.ar.ac_type(i).crews_auth.ampn.change THEN
          putxy(52,slo+i,a_mark);
        END IF;
      END IF;
      IF u.ar.ac_type(i).crews_auth.delete THEN
        putxy(50,slo+i,d_mark);
      END IF;
      putxy(54,slo+i,u.ar.ac_type(i).crews_auth.iss);
              -----
      putxy(69,slo+i,Int_to_Str(u.ar.ac_type(i).
                                    number_of_locations));
    END LOOP;
  END fill_auth_mask;

PROCEDURE process_auth(i: IN INTEGER;
                       check_location: OUT BOOLEAN) IS
  BEGIN
    modify_airauth: LOOP
      putxy(33,slo+i,blank);
      getxy_immediate(33,slo+i,key);
      CASE key IS
        WHEN no_input|space =>
          EXIT modify_airauth;
        WHEN c =>
          IF u.ar.ac_type(i).ac_auth.delete THEN
            error(31);          -- deleted/cant modify
          ELSE
```

```
        LOOP
          buffer := getxy_digits(40,slo+i,2);
          IF buffer = null_string THEN
            putxy(40,slo+i,
                         u.ar.ac_type(i).ac_auth.iss);
            EXIT;
          ELSE
            u.ar.ac_type(i).ac_auth.iss := buffer;
            u.ar.ac_type(i).ac_auth.change := TRUE;
            u.ar.ac_type(i).change := TRUE;
            putxy(36,slo+i,star);
            putxy(40,slo+i,
                         u.ar.ac_type(i).ac_auth.iss);
            EXIT;
          END IF;
        END LOOP;
      END IF;
    WHEN a =>
      IF u.ar.ac_type(i).ac_auth.change THEN
        process_comment("AIRATE"& Int_to_Str(i),
                        u.ar.ac_type(i).ac_auth.ampn);
        fill_auth_mask;
      ELSE
        error(27);          --no set to amplify
      END IF;
    WHEN d =>
      IF u.ar.ac_type(i).ac_auth.change THEN
        error(35);     -- set changed, can't delete
      ELSE
        u.ar.ac_type(i).ac_auth.delete := TRUE;
        putxy(36,slo+i,d_mark);
      END IF;
    WHEN r =>
      IF u.ar.ac_type(i).ac_auth.delete THEN
        u.ar.ac_type(i).ac_auth.delete := FALSE;
        putxy(38,slo+i,blank);
      ELSE
        error(34);          -- set not deleted
      END IF;
    WHEN ques =>
      help("AIRAUTH");
      fill_auth_mask;
    WHEN OTHERS =>
      error(6);    -- incorrect input
  END CASE;
END LOOP modify_airauth;


modify_crewauth: LOOP
  putxy(47,slo+i,blank);
  getxy_immediate(47,slo+i,key);
```

```
CASE key IS
  WHEN no_input|space =>
    EXIT modify_crewauth;
  WHEN c =>
    IF u.ar.ac_type(i).crews_auth.delete THEN
      error(31);          -- deleted/cant modify
    ELSE
      LOOP
        buffer := getxy_digits(54,slo+i,2);
        IF buffer = null_string THEN
          putxy(54,slo+i,
                    u.ar.ac_type(i).crews_auth.iss);
          EXIT;
        ELSE
          u.ar.ac_type(i).crews_auth.iss := buffer;
          u.ar.ac_type(i).crews_auth.change := TRUE;
          putxy(50,slo+i,star);
          u.ar.ac_type(i).change := TRUE;
          putxy(54,slo+i,
                    u.ar.ac_type(i).crews_auth.iss);
          EXIT;
        END IF;
      END LOOP;
    END IF;
  WHEN a =>
    IF u.ar.ac_type(i).crews_auth.change THEN
      process_comment("CRWATE"& Int_to_Str(i),
                   u.ar.ac_type(i).crews_auth.ampn);
      fill_auth_mask;
    ELSE
      error(27);          --no set to amplify
    END IF;
  WHEN d =>
    IF u.ar.ac_type(i).crews_auth.change THEN
      error(35);  -- set changed, can't delete
    ELSE
      u.ar.ac_type(i).crews_auth.delete := TRUE;
      putxy(50,slo+i,d_mark);
    END IF;
  WHEN r =>
    IF u.ar.ac_type(i).crews_auth.delete THEN
      u.ar.ac_type(i).crews_auth.delete := FALSE;
      putxy(52,slo+i,blank);
    ELSE
      error(34);          -- set not deleted
    END IF;
  WHEN ques =>
    help("CREWAUTE");
    fill_auth_mask;
  WHEN OTHERS =>
    error(6);   -- incorrect input
```

```
      END CASE;
    END LOOP modify_crewauth;

    check_location := mark(62,slo+i);
  END process_auth;

END urair2;
```

```
PACKAGE urair3 IS

     -- Interface to air status information

  slo: CONSTANT := 5;   -- screen line offset

  PROCEDURE blank_inst_area;

  PROCEDURE process_status(i,j: IN INTEGER);

END urair3;
```

```
  WITH io,strlib,                  --JANUS/ADA library
       consio,
       urgltl,
       unitrepB,
       urairB,
       urutil,
       urglobal;
PACKAGE BODY urair3 IS
  USE consio,
      urgltl,
      unitrepB,
      urairB,
      urutil,
      urglobal;

        -- This package provides the routines necessary to
        -- modify air status information.
        -- (AIRSTAT, CREWSTAT, RECON data sets)

  PROCEDURE blank_inst_area IS
    BEGIN
      FOR y IN 14..20 LOOP
        putxy(0,y,blankln);
      END LOOP;
    END blank_inst_area;

  PROCEDURE process_status(i,j: IN INTEGER) IS
    USE strlib;

    PROCEDURE recon_mask IS
      BEGIN
        putxy(8,9,"( )     -3- Reconaissance Capability   " &
                  "Primary --------");
        putxy(10,11,"--------   --------   --------    " &
                    "--------   --------");
        putxy(10,12,"--------   --------   --------    " &
                    "--------   --------");
      END recon_mask;

    PROCEDURE status_mask IS
      USE strlib;         -- Extract
      BEGIN
        border;
        putxy(0,1,"AIR - STATUS");
        putxy(0,3,"Aircraft Type -------");
        putxy(31,3,"( )   Location");
        putxy(48,3,Extract(dashln,1,30));
        putxy(6,5,"( )     -1- Aircraft");
        putxy(48,5,"( )     -2- Crews");
        putxy(2,6,"Possessed    FMC   PMC   NMC");
        putxy(48,6,"Formed      Ready");
```

```
    putxy(5,7,"--"); putxy(16,7,"--     --     --");
    putxy(50,7,"--          --");
    recon_mask;
    putxy(0,13,dashln);
    putxy(0,14,"ENTER   <C> - to CHANGE -  " &
               "(in fields 1,2,3 only)");
    putxy(8,15,"<D> - to DELETE -  (at Location will " &
               "DELETE ALL information)");
    putxy(8,16,
         "<R> - to RESTORE - (restores deleted fields)");
    putxy(8,17,
         "<A> - to AMPLIFY - (in fields 1,2,3 only)");
    putxy(8,18,"<?> - for HELP   - (all fields)");
    putxy(0,19,"THEN: ENTER Required information.");
    putxy(8,20,"(an <ENTER> with no information is " &
               "assurred to be NO CHANGE)");
  END status_mask;

PROCEDURE fill_recon IS
  BEGIN
    recon_mask;
    putxy(54,9,
       u.ar.ac_type(i).location(j).recon.capability(1));
    FOR k IN 2..6 LOOP
      putxy(10+(k-2)*11,11,
       u.ar.ac_type(i).location(j).recon.capability(k));
    END LOOP;
    FOR k IN 7..11 LOOP
      putxy(10+(k-7)*11,12,
       u.ar.ac_type(i).location(j).recon.capability(k));
    END LOOP;
    IF u.ar.ac_type(i).location(j).recon.delete THEN
      putxy(12,9,d_mark);
    ELSIF u.ar.ac_type(i).location(j).recon.change THEN
      putxy(12,9,star);
      IF u.ar.ac_type(i).location(j).recon.ampn.change
                                                     THEN
        putxy(14,9,a_mark);
      END IF;
    END IF;
  END fill_recon;


PROCEDURE fill_status_mask IS
  BEGIN
    status_mask;
    putxy(14,3,u.ar.ac_type(i).iss);
    putxy(48,3,u.ar.ac_type(i).location(j).iss);
    IF j = 1 THEN
      putxy(48,3,"HOME PLATE");
    END IF;
```

```
    IF u.ar.ac_type(i).location(j).delete THEN
      putxy(35,3,d_mark);
    ELSIF u.ar.ac_type(i).location(j).change THEN
      putxy(35,3,star);
    END IF;
                    -----
    IF u.ar.ac_type(i).location(j).airstat.delete THEN
      putxy(10,5,d_mark);
    ELSIF u.ar.ac_type(i).location(j).airstat.change
                                                   THEN
      putxy(10,5,star);
      IF u.ar.ac_type(i).location(j).
                                airstat.ampn.change THEN
        putxy(12,5,a_mark);
      END IF;
    END IF;
    putxy(5,7,
         u.ar.ac_type(i).location(j).airstat.possessed);
    putxy(16,7,u.ar.ac_type(i).location(j).airstat.fmc);
    putxy(22,7,u.ar.ac_type(i).location(j).airstat.pmc);
    putxy(28,7,u.ar.ac_type(i).location(j).airstat.nmc);
                    -----
    IF u.ar.ac_type(i).location(j).crewstat.delete THEN
      putxy(52,5,d_mark);
    ELSIF u.ar.ac_type(i).location(j).crewstat.change
                                                   THEN
      putxy(52,5,star);
      IF u.ar.ac_type(i).location(j).
                               crewstat.ampn.change THEN
        putxy(54,5,a_mark);
      END IF;
    END IF;
    putxy(50,7,
           u.ar.ac_type(i).location(j).crewstat.formed);
    putxy(62,7,
            u.ar.ac_type(i).location(j).crewstat.ready);
                    -----
    fill_recon;
  END fill_status_mask;

PROCEDURE process_recon(index,x,y: IN INTEGER) IS
    -- processes a single recon set

  USE io;   --Open,CLOSE,GET_LINE
  USE strlib; -- Extract,Position,Char_to_Str,Int_to_Str
  BEGIN
    getxy_immediate(x,y,key);
    CASE key IS
      WHEN no_input =>
        NULL;
      WHEN space => -- delete set move others up
```

```
  FOR k IN index..max_recon-1 LOOP
      u.ar.ac_type(i).location(j).
                            recon.capability(k) :=
      u.ar.ac_type(i).location(j).
                            recon.capability(k+1);
  END LOOP;
  u.ar.ac_type(i).location(j).
       recon.capability(max_recon) := null_string;
  u.ar.ac_type(i).location(j).
                             recon.change := TRUE;
  IF u.ar.ac_type(i).location(j).
             recon.number_of_capabilities > 0 THEN
    u.ar.ac_type(i).location(j).
                   recon.number_of_capabilities :=
    u.ar.ac_type(i).location(j).
                 recon.number_of_capabilities - 1;
  END IF;
  fill_recon;
WHEN OTHERS =>        -- add new recon to the set
  IF u.ar.ac_type(i).location(j).
    recon.number_of_capabilities >= max_recon THEN
    error(40); -- max sets, delete first then add
  ELSE
    FOR k IN REVERSE index+1..max_recon LOOP
      u.ar.ac_type(i).location(j).
                            recon.capability(k) :=
      u.ar.ac_type(i).location(j).
                            recon.capability(k-1);
    END LOOP;
    u.ar.ac_type(i).location(j).
           recon.capability(index) := null_string;
    fill_recon;
    buffer := Char_to_Str(key);
    putxy(x,y,buffer);
    getxy(x+1,y,file_buffer,7);
    buffer := buffer & file_buffer;
    Open(fyle,"TABLE.B-3",Read_Only);
    LOOP
      file_buffer := GET_LINE(fyle);
      file_buffer :=
              Extract(file_buffer,1,
                    Position(":",file_buffer) -1);
      IF file_buffer = buffer THEN
        CLOSE(fyle);
        u.ar.ac_type(i).location(j).
                             recon.change := TRUE;
        u.ar.ac_type(i).location(j).
                recon.capability(index) := buffer;
        u.ar.ac_type(i).location(j).
                   recon.number_of_capabilities :=
        u.ar.ac_type(i).location(j).
```

```
                       recon.number_of_capabilities + 1;
              EXIT;
            END IF;
            IF END_OF_FILE(fyle) THEN
              CLOSE(fyle);
              error(41);    -- invalid recon capability
              putxy(x,y,"--------");
              EXIT;
            END IF;
          END LOOP;
        END IF;
      END CASE;
            -- clean up null strings due to
            -- faulty entries/deletions
          FOR k IN 1..max_recon LOOP
            IF u.ar.ac_type(i).location(j).
                  recon.capability(k) = null_string THEN
              FOR l IN k..max_recon-1 LOOP
                u.ar.ac_type(i).location(j).
                                  recon.capability(l) :=
                    u.ar.ac_type(i).location(j).
                                 recon.capability(l+1) ;
              END LOOP;
             u.ar.ac_type(i).location(j).
             recon.capability(max_recon) := null_string;
            END IF;
          END LOOP;
        END process_recon;


BEGIN
  modify_status: LOOP
    fill_status_mask;

    check_location: LOOP
      putxy(32,3,blank);
      getxy_immediate(32,3,key);
      CASE key IS
        WHEN no_input|space =>
          EXIT check_location;
        WHEN d =>
          IF u.ar.ac_type(i).location(j).change OR
             u.ar.ac_type(i).location(j).
                                        airstat.change OR
             u.ar.ac_type(i).location(j).
                                       crewstat.change OR
             u.ar.ac_type(i).location(j).
                                        recon.change THEN
            error(35);      -- changed, can't delete
          ELSE
            u.ar.ac_type(i).location(j).delete := TRUE;
```

```
        u.ar.ac_type(i).location(j).
                             airstat.delete := TRUE;
        u.ar.ac_type(i).location(j).
                            crewstat.delete := TRUE;
        u.ar.ac_type(i).location(j).
                               recon.delete := TRUE;
        putxy(35,3,d_mark);
        putxy(10,5,d_mark);
        putxy(52,5,d_mark);
        putxy(12,9,d_mark);
      END IF;
    WHEN r =>
      IF u.ar.ac_type(i).location(j).delete OR
         u.ar.ac_type(i).location(j).
                                   airstat.delete OR
         u.ar.ac_type(i).location(j).
                                  crewstat.delete OR
         u.ar.ac_type(i).location(j).
                                   recon.delete THEN

        u.ar.ac_type(i).location(j).delete := FALSE;
        u.ar.ac_type(i).location(j).
                            airstat.delete := FALSE;
        u.ar.ac_type(i).location(j).
                           crewstat.delete := FALSE;
        u.ar.ac_type(i).location(j).
                              recon.delete := FALSE;
        putxy(35,3,blank);
        putxy(10,5,blank);
        putxy(52,5,blank);
        putxy(12,9,blank);
      ELSE
        error(34);      -- not deleted
      END IF;
    WHEN ques =>
      help("LOCATION");
      fill_status_mask;
    WHEN OTHERS =>
      error(6);         -- incorrect input
  END CASE;
END LOOP check_location;

modify_airstat: LOOP
  putxy(7,5,blank);
  getxy_immediate(7,5,key);
  CASE key IS
    WHEN no_input|space =>
      EXIT modify_airstat;
    WHEN c =>
      IF u.ar.ac_type(i).location(j).
                                 airstat.delete THEN
```

```
  error(31);        -- cant modify deleted
ELSE
  LOOP
    buffer := getxy_digits(5,7,2);
    EXIT WHEN buffer = null_string;
    u.ar.ac_type(i).location(j).
                  airstat.possessed := buffer;
    u.ar.ac_type(i).location(j).
                       airstat.change := TRUE;
    putxy(10,5,star);
    u.ar.ac_type(i).location(j).change :=TRUE;
    EXIT;
  END LOOP;
  putxy(5,7,u.ar.ac_type(i).location(j).
                           airstat.possessed);
  LOOP
    buffer := getxy_digits(16,7,2);
    EXIT WHEN buffer = null_string;
    u.ar.ac_type(i).location(j).
                        airstat.fmc := buffer;
    u.ar.ac_type(i).location(j).
                       airstat.change := TRUE;
    putxy(10,5,star);
    u.ar.ac_type(i).location(j).change :=TRUE;
    EXIT;
  END LOOP;
  putxy(16,7,u.ar.ac_type(i).location(j).
                                 airstat.fmc);
  LOOP
    buffer := getxy_digits(22,7,2);
    EXIT WHEN buffer = null_string;
    u.ar.ac_type(i).location(j).
                        airstat.pmc := buffer;
    u.ar.ac_type(i).location(j).
                       airstat.change := TRUE;
    putxy(10,5,star);
    u.ar.ac_type(i).location(j).change :=TRUE;
    EXIT;
  END LOOP;
  putxy(22,7,u.ar.ac_type(i).location(j).
                                 airstat.pmc);
  LOOP
    buffer := getxy_digits(28,7,2);
    EXIT WHEN buffer = null_string;
    u.ar.ac_type(i).location(j).
                        airstat.nmc := buffer;
    u.ar.ac_type(i).location(j).
                       airstat.change := TRUE;
    putxy(10,5,star);
    u.ar.ac_type(i).location(j).change :=TRUE;
    EXIT;
```

```
        END LOOP;
        putxy(28,7,u.ar.ac_type(i).location(j).
                                        airstat.nmc);
        IF Str_to_Int(u.ar.ac_type(i).location(j).
                                airstat.possessed) /=
             Str_to_Int(u.ar.ac_type(i).location(j).
                                       airstat.fmc) +
             Str_to_Int(u.ar.ac_type(i).location(j).
                                       airstat.pmc) +
             Str_to_Int(u.ar.ac_type(i).location(j).
                                   airstat.nmc)  THEN
          error(38);
                -- possessd/fmc.pmc.nmc don't sum
        END IF;
      END IF;
    WHEN a =>
      IF u.ar.ac_type(i).location(j).airstat.change
                                                 THEN
        process_comment("AIRSTA"& Int_to_Str(i) &
                          Int_to_Str(j),
           u.ar.ac_type(i).location(j).airstat.ampn);
        fill_status_mask;
      ELSE
        error(27);      --not changed
      END IF;
    WHEN d =>
      IF u.ar.ac_type(i).location(j).airstat.change
                                                 THEN
        error(35);      --set modified/can't change
      ELSE
        u.ar.ac_type(i).location(j).
                             airstat.delete := TRUE;
        putxy(10,5,d_mark);
                -- also delete recon
        IF u.ar.ac_type(i).location(j).
                    recon.number_of_capabilities > 0
          THEN u.ar.ac_type(i).location(j).
                                recon.delete := TRUE;
          putxy(12,9,d_mark);
        END IF;
      END IF;
    WHEN r =>
      IF u.ar.ac_type(i).location(j).airstat.delete
                                                 THEN
        u.ar.ac_type(i).location(j).
                            airstat.delete := FALSE;
        putxy(10,5,blank);
      ELSE
        error(34);  -- not deleted/can't restore
      END IF;
    WHEN ques =>
```

```
      help("AIRSTAT");
      fill_status_mask;
    WHEN OTHERS =>
      error(6);         -- in correct input
  END CASE;
END LOOP modify_airstat;

modify_crewstat: LOOP
  putxy(49,5,blank);
  getxy_immediate(49,5,key);
  CASE key IS
    WHEN no_input|space =>
      EXIT modify_crewstat;
    WHEN c =>
      IF u.ar.ac_type(i).location(j).crewstat.delete
                                                THEN
        error(31);      -- cant modify deleted
      ELSE
        LOOP
          buffer := getxy_digits(50,7,2);
          EXIT WHEN buffer = null_string;
          u.ar.ac_type(i).location(j).
                          crewstat.formed := buffer;
          u.ar.ac_type(i).location(j).
                            crewstat.change := TRUE;
          putxy(52,5,star);
          u.ar.ac_type(i).location(j).change :=TRUE;
          EXIT;
        END LOOP;
        putxy(50,7,u.ar.ac_type(i).location(j).
                                   crewstat.formed);
        LOOP
          buffer := getxy_digits(62,7,2);
          EXIT WHEN buffer = null_string;
          u.ar.ac_type(i).location(j).
                           crewstat.ready := buffer;
          u.ar.ac_type(i).location(j).
                            crewstat.change := TRUE;
          putxy(52,5,star);
          u.ar.ac_type(i).location(j).change :=TRUE;
          EXIT;
        END LOOP;
        putxy(62,7,u.ar.ac_type(i).location(j).
                                    crewstat.ready);
        IF Str_to_INT(u.ar.ac_type(i).location(j).
                                  crewstat.formed) <
             Str_to_Int(u.ar.ac_type(i).location(j).
                                crewstat.ready) THEN
          error(39);    --crews ready > crews formed
        END IF;
      END IF;
```

```
    WHEN a =>
      IF u.ar.ac_type(i).location(j).
                                crewstat.change THEN
        process_comment("CRWSTA" & Int_to_Str(i) &
          Int_to_Str(j),u.ar.ac_type(i).location(j).
                                     crewstat.ampn);
        fill_status_mask;
      ELSE
        error(27);      -- no set to amplify
      END IF;
    WHEN d =>
      IF u.ar.ac_type(i).location(j).crewstat.change
                                                 THEN
        error(35);
      ELSE
        u.ar.ac_type(i).location(j).
                            crewstat.delete := TRUE;
        putxy(52,5.d_mark);
      END IF;
    WHEN r =>
      IF u.ar.ac_type(i).location(j).
                               crewstat.delete THEN
        u.ar.ac_type(i).location(j).
                           crewstat.delete := FALSE;
        putxy(52,5,blank);
      ELSE
        error(34);      --not deleted/can´t restore
      END IF;
    WHEN ques =>
      help("CREWSTAT");
      fill_status_mask;
    WHEN OTHERS =>
      error(6);         -- incorrect input
  END CASE;
END LOOP modify_crewstat;

modify_recon: LOOP
  putxy(9,9,blank);
  getxy_immediate(9.9,key);
  CASE key IS
    WHEN no_input|space =>
      EXIT modify_recon;
    WHEN c =>
      IF u.ar.ac_type(i).location(j).recon.delete
                                                 THEN
        error(31);      -- cant modify deleted
      ELSE
        putxy(12,9.star);
        blank_inst_area;
        putxy(8,15,"FOR RECON CHANGES...");
        putxy(8.16,"<ENTER> - go to next item");
```

```
        putxy(8,17,"space   - erase this item");
        putxy(8,18,
           "all others - add item at this location");
        process_recon(1,54,9);
        FOR k IN 2..6 LOOP
          process_recon(k,10+(k-2)*11,11);
        END LOOP;
        FOR k IN 7..11 LOOP
          process_recon(k,10+(k-7)*11,12);
        END LOOP;
      END IF;
    WHEN a =>
      IF u.ar.ac_type(i).location(j).recon.change
                                                 THEN
        process_comment("RECON" & Int_to_Str(i) &
          Int_to_Str(j),u.ar.ac_type(i).location(j).
                                         recon.ampn);
        fill_status_mask;
      ELSE
        error(27); -- set not changed/can't amplify
      END IF;
    WHEN d =>
      IF u.ar.ac_type(i).location(j).recon.change
                                                 THEN
        error(35);      -- cant delete modified set
      ELSE
        u.ar.ac_type(i).location(j).
                                recon.delete := TRUE;
        putxy(12,9,d_mark);
      END IF;
    WHEN r =>
      IF u.ar.ac_type(i).location(j).recon.delete
                                                 THEN
        u.ar.ac_type(i).location(j).
                               recon.delete := FALSE;
        putxy(12,9,blank);
      ELSE
        error(34);      -- not deleted/cant restore
      END IF;
    WHEN ques =>
      help("RECON");
      view_table("TABLE.B-3");
      fill_status_mask;
    WHEN OTHERS =>
      error(6);         -- invalid input
  END CASE;
END LOOP modify_recon;
blank_inst_area;
IF u.ar.ac_type(i).location(j).delete OR
   u.ar.ac_type(i).location(j).airstat.delete OR
   u.ar.ac_type(i).location(j).crewstat.delete OR
```

```
         u.ar.ac_type(i).location(j).recon.delete THEN
         PUT(bell);
         putxy(13,16,"Data is to be deleted. Ensure " &
                     "that the corresponding");
         putxy(8,17,"data items are adjusted in another" &
                    "STATUS dislay. (if required)");
         putxy(30,19,"<ENTER> to Continue ");
         getxy_immediate(52,19,key);
      END IF;
      blank_inst_area;
      putxy(0,17,"<ENTER> to Continue, Enter <X> to " &
                 "Reselect:");
      reselect := mark(44,17);
      EXIT modify_status WHEN NOT reselect;
    END LOOP modify_status;
  END process_status;

END urair3;
```

```
  WITH io,              -- JANUS/ADA Library
       printio,
       urgltl,
       urglobal,
       urlocalA,
       unitrepA;
PACKAGE BODY initialA IS
  USE io,               -- Open,CLOSE,DELETE
      printio,
      urgltl,
      urglobal,
      urlocalA,
      unitrepA;

        -- This package is used to form:
        -- the initial workfile (UNIT000A),
                -- elements of the local data structure
        -- the initial program status file (STATUS),
        -- and the initial cross reference file (CROSSREF).
        -- When urtest is incorporated the test file
        -- TEST000 is also formed.

        -- The INITIALA.COM file resulting from this
        -- package should not be distributed to all users.
        -- Only those organizations required to set the
        -- UNIT000A, STATUS and CROSSREF external files
        -- should have access to this code.

        -- UNIT000A, STATUS and CROSSREF files are added
        -- to the originating units UNITREP MASTER DISK after
        -- initialization has been performed.


  fyle: FILE;
  serial_dtg: STRING(16);
--#  test_number: INTEGER;

  BEGIN
        --    initialize test data - sets A:TEST000 to 0
--#      DELETE("TEST000");
--#      PUT("Creating new TEST000..."); NEW_LINE;
--#      Create(fyle,"TEST000",Write_Only);
--#      Open(fyle,"TEST000",Write_Only);
--#      test_number := 0;
--#      PUT(fyle,test_number);
--#      CLOSE(fyle);
--#      PUT("Put of 0 to TEST000 complete... ");
--#      NEW_LINE; NEW_LINE;
```

```
              -- initialize the program status
program.unit := ship;
program.initial_entry := TRUE;
program.print_trailer := FALSE;
program.message_transmitted := FALSE;
FOR i IN action RANGE draft..quit LOOP
  program.current_action(i) := FALSE;
END LOOP;
FOR i IN areas RANGE pers..local  LOOP
  program.current_area(i) := FALSE;
END LOOP;
FOR i IN format RANGE msg..text LOOP
  program.current_format(i) := FALSE;
END LOOP;
program.printing_line := 1;
program.workfile := "UNIT000";
DELETE("STATUS");
Create(fyle,"STATUS",Write_Only);
Write(fyle,program);
CLOSE(fyle);
PUT("Write to STATUS complete...");
NEW_LINE; NEW_LINE;

-- initialize UNIT000
u.l.change := FALSE;
u.l.status.transmitted := FALSE;
u.l.status.last_serial := "000";
u.l.status.feeder_report := FALSE;
u.l.status.unit_id := "";
u.l.status.current_serial := "001";
u.l.status.ampn.change := FALSE;
u.l.status.ampn.number_of_lines := 0;

u.l.message.precedence := priority;
u.l.message.dtg := "";
u.l.message.originator_address := "";
u.l.message.classification := confidential;
u.l.message.declassification := "";

u.l.operation.underway := FALSE;
u.l.operation.codeword := "";
u.l.operation.plan_org_number := "";

u.l.exercise.underway := FALSE;
u.l.exercise.nickname := "";

u.l.position.change := FALSE;
u.l.position.lat_long := FALSE;
u.l.position.present_location := "";
u.l.position.dtg := "";
u.l.position.ampn.change := FALSE;
```

```
  u.l.position.ampn.number_of_lines := 0;

  u.l.remarks.change := FALSE;
  u.l.remarks.number_of_pages := 0;
  FOR i IN 1..max_rmks_pages LOOP
    u.l.remarks.page(i).change := FALSE;
    u.l.remarks.page(i).number_of_lines := 0;
  END LOOP;



  DELETE("UNIT000A");
  Create(fyle,"UNIT000A",Write_Only);
  Open(fyle,"UNIT000A",Write_Only);
  Write(fyle,u);
  CLOSE(fyle);
  PUT("Write to UNIT000A Complete...");
  NEW_LINE; NEW_LINE;

  -- set up the cross reference file
  serial_dtg := "000 ddhhmmZMMMyy";
  DELETE("CROSSREF");
  Create(fyle,"CROSSREF",Write_Only);
  PUT(fyle,serial_dtg); NEW_LINE(fyle);
  CLOSE(fyle);
  PUT("PUT to CROSSREF Complete...");
  NEW_LINE; NEW_LINE;

END initialA;
```

```
  WITH io,        -- JANUS/ADA Library
       urgltl,
       uradminB,
       urairB,
       unitrepB;
PACKAGE BODY initialB IS
  USE io;
  USE urgltl;
  USE urairB;
  USE uradminB;
  USE unitrepB;
          -- initialize the 'B' data structure

  two0: STRING;
  fyle: FILE;

  PROCEDURE zero_location(i,j: IN INTEGER) IS
    BEGIN
                -- zero location set (i,j)
      u.ar.ac_type(i).location(j).change := FALSE;
      u.ar.ac_type(i).location(j).delete := FALSE;
      IF j = 1 THEN
                -- present position
        u.ar.ac_type(i).location(j).iss := "-";
      ELSE
        u.ar.ac_type(i).location(j).iss := "";
      END IF;
      u.ar.ac_type(i).location(j).airstat.change := FALSE;
      u.ar.ac_type(i).location(j).airstat.delete := FALSE;
      u.ar.ac_type(i).location(j).airstat.possessed := two0;
      u.ar.ac_type(i).location(j).airstat.fmc := two0;
      u.ar.ac_type(i).location(j).airstat.pmc := two0;
      u.ar.ac_type(i).location(j).airstat.nmc := two0;
      u.ar.ac_type(i).location(j).
                               airstat.ampn.change := FALSE;
      u.ar.ac_type(i).location(j).
                          airstat.ampn.number_of_lines := 0;
      u.ar.ac_type(i).location(j).crewstat.change := FALSE;
      u.ar.ac_type(i).location(j).crewstat.delete := FALSE;
      u.ar.ac_type(i).location(j).crewstat.formed := two0;
      u.ar.ac_type(i).location(j).crewstat.ready := two0;
      u.ar.ac_type(i).location(j).
                              crewstat.ampn.change := FALSE;
      u.ar.ac_type(i).location(j).
                         crewstat.ampn.number_of_lines := 0;
      u.ar.ac_type(i).location(j).recon.change := FALSE;
      u.ar.ac_type(i).location(j).recon.delete := FALSE;
      u.ar.ac_type(i).location(j).recon.
                                number_of_capabilities := 0;
      u.ar.ac_type(i).location(j).
                                 recon.ampn.change := FALSE;
```

```
    u.ar.ac_type(i).location(j).
                          recon.ampn.number_of_lines := 0;
    FOR z IN 1..max_recon LOOP
      u.ar.ac_type(i).location(j).
                                 recon.capability(z) := "";
   END LOOP;
 END zero_location;


PROCEDURE zero_ac_type(i: IN INTEGER) IS
  BEGIN
              -- zero out all items
    u.ar.ac_type(i).change := FALSE;
    u.ar.ac_type(i).delete := FALSE;
    u.ar.ac_type(i).iss := "";
    u.ar.ac_type(i).ac_auth.change := FALSE;
    u.ar.ac_type(i).ac_auth.delete := FALSE;
    u.ar.ac_type(i).ac_auth.iss := two0;
    u.ar.ac_type(i).ac_auth.ampn.change := FALSE;
    u.ar.ac_type(i).ac_auth.ampn.number_of_lines := 0;
    u.ar.ac_type(i).crews_auth.change := FALSE;
    u.ar.ac_type(i).crews_auth.delete := FALSE;
    u.ar.ac_type(i).crews_auth.iss := two0;
    u.ar.ac_type(i).crews_auth.ampn.change := FALSE;
    u.ar.ac_type(i).crews_auth.ampn.number_of_lines := 0;
    u.ar.ac_type(i).number_of_locations := 1; -- homeplate
    FOR j IN 1..max_locations LOOP
      zero_location(i,j);
    END LOOP;
  END zero_ac_type;


BEGIN
  two0 := "00";

  -- initialize UNIT000B

              -- admin sets
  u.ad.change := FALSE;

  u.ad.command.change := FALSE;
  u.ad.command.cooic := "";
  u.ad.command.dtg := "";
  u.ad.command.ampn.change := FALSE;
  u.ad.command.ampn.number_of_lines := 0;

  u.ad.activ.change := FALSE;
  u.ad.activ.activity_code := "CP";
  u.ad.activ.ampn.change := FALSE;
  u.ad.activ.ampn.number_of_lines := 0;
```

```
  u.ad.medic.change := FALSE;
  u.ad.medic.status := onboard;
  u.ad.medic.ampn.change := FALSE;
  u.ad.medic.ampn.number_of_lines := 0;

  u.ad.reporg.change := FALSE;
  u.ad.reporg.new_rep_org := cno;
  u.ad.reporg.dtg := "";
  u.ad.reporg.ampn.change := FALSE;
  u.ad.reporg.ampn.number_of_lines := 0;

  u.ad.verify.change := FALSE;
  u.ad.verify.feedback_ver := valid;
  u.ad.verify.ampn.change := FALSE;
  u.ad.verify.ampn.number_of_lines := 0;

               -- air sets
  u.ar.change := FALSE;
  u.ar.number_of_ac_types := 0;
  FOR i IN 1..max_ac_types LOOP
    zero_ac_type(i);
  END LOOP;

  DELETE("UNIT000B");
  Create(fyle,"UNIT000B",Write_Only);
  Open(fyle,"UNIT000B",Write_Only);
  Write(fyle,u);
  CLOSE(fyle);
  PUT("Write to UNIT000B Complete...");
  NEW_LINE; NEW_LINE;

END initialB;
```

LIST OF REFERENCES

1. Naval Electronics Systems Command, *Test and Evaluation Report X/C 13 Increment I (COMPREP)*, 30 June 1975.

2. Naval Electronics Systems Command Report PME-108-P00011. *Fleet Command Center Composite Operations Reporting System*, 20 August 1976.

3. Holyoak, J.G., *A Shipboard Report Origination System Utilizing a Microcomputer*, M.S. Thesis, Naval Postgraduate School, Monterey, CA., 1976.

4. Godley, J.B., *A Microcomputer Based Generator of Recurring Operational Reports*, M.S. Thesis, Naval Postgraduate School, Monterey, CA., 1977.

5. Office of the Chief of Naval Operations proposed Instruction 3503.1 - 3503.7, *Navy Reporting Structure Operational Reports*, 15 January 1982.

6. Naval Electronics Systems Command, *Navy Command and Control System (NCCS) Formatted Message Error Analysis Report*, Sterling Data Applications, Inc., Sterling, VA., 1 October 1980.

7. Sterling Data Applications Technical Report 0617-D13, *Result of Composite Operational Reporting System Report Origination System Prototype Development Test*, 12 March 1982.

8. *Ada Programming Language*, Department of Defense Military Standard MIL-STD-1815, 10 December 1980.

9. *JANUS/ADA Package User Manual 8080 Version 3*, RR Software, Madison, WI., July 1982.

10. Office of the Chief of Naval Operations proposed Instruction 3503.5 *Navy Reporting Structure Operational Reports - Unit Status and Identity Report*, 15 January 1982.

BIBLIOGRAPHY

Barnes,J.G.P., Programming in ADA, Addison-Wesley Publishing Company, 1982.

Sprague Jr., Ralph H. and Carlson, Eric D., Building Effective Decision Support Systems, Prentice-Hall Inc., 1982.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3.	Naval Electronics Systems Command PME-120-1213 National Center 1 (Room 8E72) Washington, D.C. 20363	2
4.	Prof. Hal Titus (Code 62 Ts) Department of Electrical Engineering Naval Postgraduate School Monterey, California 93940	5
5.	CDR Gary R. Porter (Code 55 Pt) Director, Command and Control War Laboratory Naval Postgraduate School Monterey, California 93940	1
6.	Prof. U.R. Kodres (Code 52 Kr) Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
7.	Capt. T.F. Rogers SMC 2237 Naval Postgraduate School Monterey, California 93940	1
8.	Cpt. J.A. Karadimitropoulos SMC 2519 Naval Postgraduate School Monterey, California 93940	1
9.	LT. M.R. Critz HSL-33 NAS North Island San Diego, California 92135	1

Thesis
C8735
c.1

200677

Critz

An approach for implementing a micro-computer based report origination system in the Ada programming language.

30 AUG 84
30 AUG 84 29740
10 DEC 86 29740
16 AUG 89 33324
80063

Thesis
C8735
c.1

200677

Critz

An approach for implementing a micro-computer based report origination system in the ADA programming language.

thesC8735
An approach for implementing a microcomp
3 2768 001 02467 2
DUDLEY KNOX LIBRARY